Stanford Artificial Intelligence Laboratory
Memo AIM-328

Computer Science Department
Report No. STAN-CS-79-743

June 1979

GOAL:
A GOAL ORIENTED COMMAND LANGUAGE FOR INTERACTIVE PROOF CONSTRUCTION

by

Juan Bautista Buines-Rozas

Research sponsored by

Advanced Research Projects Agency

COMPUTER SCIENCE DEPARTMENT
Stanford University

Stanford Artificial Intelligence Laboratory
Memo AIM-328

June 1979

Computer Science Department
Report No. STAN-CS-79-743

GOAL:
A GOAL ORIENTED COMMAND LANGUAGE FOR INTERACTIVE PROOF CONSTRUCTION

by

Juan Bautista Buines-Rozas

ABSTRACT

This thesis represents a contribution to the development of practical computer systems for interactive construction of formal proofs. Beginning with a summary of current research in automatic theorem proving, goal oriented systems, proof checking, and program verification, this work aims at bridging the gap between proof checking and theorem proving.

Specifically, it describes a system GOAL for the First Order Logic proof checker FOL. GOAL helps the user of FOL in the creation of long proofs in three ways: 1) as a facility for structured, top down proof construction; 2) as a semi-automatic theorem prover; and 3) as an extensible environment for the programming of theorem proving heuristics.

In GOAL, the user defines top level goals. These are then recursively decomposed into subgoals. The main part of a goal is a well formed formula that one desires to prove, but they include assertions, simplification sets, and other information. Goals can be tried by three different types of elements: *matchers*, *tactics*, and *strategies*.

The *matchers* attempt to prove a goal directly -that is without reducing it into subgoals- by calling decision procedures of FOL. Successful application of a matcher causes the proved goal to be added to the FOL proof.

A *tactic* reduces a goal into one or more subgoals. Each tactic is the inverse of some inference rule of FOL; the goal structure records all the necessary information so that the appropriate inference rule is called when all the subgoals of a goal are proved. In this way the goal tree unwinds automatically, producing a FOL proof of the top level goal from the proofs or its leaves.

The *strategies* are programmed sequences of applications of tactics and matchers. They do not interface with FOL directly. Instead, they simulate a virtual user of FOL. They can call the tactics, matchers, other strategies, or themselves recursively. The success of this approach to theorem proving success is documented by one heuristic strategy that has proved a number of theorems in Zermelo-Fraenkel Axiomatic Set Theory. Analysis of this strategy leads to a discussion of some trade offs related to the use of *assertions* and *simplification sets* in goal oriented theorem proving.

The user can add new tactics, matchers, and strategies to GOAL. These additions cause the language to be extended in a uniform way. The description of new strategies is done easily, at a fairly high level, and no faulty deduction is possible. Perhaps the main contribution of GOAL is a high level environment for easy programming of new theorem proving applications in the First Order Predicate Calculus.

The thesis ends with two appendixes presenting complete proofs of Ramsey's theorem in axiomatic Set Theory and of the correctness of the Takeuchi function.

(It is planned that both FOL and GOAL will be made available over the ARPANET this year. Inquiries regarding their use should be addressed to Dr. R. Weyhrauch at the Stanford Artificial Intelligence Laboratory, SU-AI).

This thesis was submitted to the Department of Computer Science and the Committee on Graduate Studies of Stanford University in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

This research was supported by the Advanced Research Projects Agency of the Department of Defense under ARPA Order No. 2494, Contract MDA903-76-C-0206. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of Stanford University, or any agency of the U. S. Government.

ACKNOWLEDGEMENTS

I am especially grateful to my adviser, John McCarthy, for his patient reading, advice, and support while this research was being carried out. I am also deeply indebted to Richard Weyhrauch for his guidance.

The form of this thesis has been markedly influenced by the criticism of Fernando Flores, Jinnet Fowles, and Bill Scherlis, as well as by the PUB macros kindly provided by Juan Ludlow.

There are many other people who have contributed directly and indirectly to the completion of this work. Among them I would like to express my special thanks to Pat Suppes for carefully reading the thesis, to my former adviser Jerry Feldman, to my teachers Volker Strassen and Don Knuth, to Les Earnest, to Herschie Allen, and to the staff of the Stanford Artificial Intelligence Laboratory, to Denny Brown, Carolyn Tajnai, and the staff of the Computer Science Department; and to the other members of the Formal Reasoning Group, in particular Andrew Robinson, Bill Glassmire, Bob Filman, Carolyn Talcott, and Chris Goad. All of these people have gone well beyond their duties in helping create a supportive and stimulating environment which made this thesis possible.

I would also like to extend my thanks to my colleagues working with **AUTOMATH** and **EXCHECK** for their conversations and hospitality.

This work has been partially supported by ARPA and the National Science Foundation.

I wish to dedicate this thesis to my parents, Juan and Sara, and to acknowledge my family and closest friends.

Table of Contents

Table of Contents

Table of Contents

1. INTRODUCTION.

The research presented in this doctoral thesis is a contribution to the development of practical systems for interactive construction of mathematical proofs.

The availability of fully interactive proof checkers that permit their users to construct proofs incrementally, gives rise to an activity which is best described by the term *interactive proof construction*. This name has not yet found widespread usage in the computer science literature; instead, related research has generally been classified into the following categories: proof checking, automatic theorem proving, and man-machine systems for these tasks. This research is related to but takes a different approach from that of previous research in those areas.

Proof checkers generally embody a system of logic that includes both the recognition of legal expressions in that logic, or *well formed formulae* (*WFF*s), and inference rules by which new formulae are deduced from axioms and/or previously proved formulae.

In an interactive proof constructor, the inference rules are embodied in commands that can be called by the user in order to increment a proof; normally, one new step of the proof is produced by every successful call to an inference command. This leads to a *bottom up* mode of proof construction, in contrast with the rather *goal oriented* thinking process of the working mathematician.

The approach taken in this thesis is to provide users of an interactive proof constructor with a language in which goals can be stated and reduced recursively into sub-goals, so that the reduction rules correspond to the inference rules of the proof constructor. Thus the goal commands are the inverse of the inference commands, and the system knows how to deduce a goal from its sub-goals. This leads to a *top down* mode of proof construction.

When an interactive proof constructor is provided with an equally interactive goal oriented command language, both modes of proof construction, the *inference oriented*, bottom up mode and the *goal oriented*, top down one, can be combined to any desired extent by the user, according to the particular problem and taste.

A novel approach to automatic theorem proving consists in replacing the human user by a heuristic for sequencing the recursive application of the goal oriented commands and of some inference commands that attempt to prove the sub-goals by using a set of *facts* or axioms. Automatic proofs of a number of theorems, including the first 33 theorems in the Appendix on Set Theory in [Kelley 1955], have been obtained with one heuristic of this type.

When a goal command language is designed to allow for easy addition of such theorem proving routines, it results in a high level programming environment for theorem proving applications. Users can program their own heuristics to fit different styles of proof and imbed them into the system without having to modify its structure. This can be done easily: the algorithms can be described as programmed sequences of calls to the reduction rules and inference commands, and priority queues or any other data structures can be used to control the order in which sub-goals are tackled. Thus users can augment the power of the

interactive proof constructor for specific domains of their interest by having their own libraries of heuristics that can be added to the system and called using the already existing high level commands of the language.

The goal oriented command language GOAL for the system FOL, an interactive proof constructor for the first order predicate calculus, is presented in this thesis. It has been programmed by the author in LISP, on top of FOL, at the Stanford Artificial Intelligence Laboratory DEC KL-10 computer system. The user can program new inference rules, new subgoaling commands, and new heuristic *strategies* as programmed sequences of calls to the inference rules and subgoaling commands; these are added to the system by calling a GOAL routine that automatically extends the language, and its syntax, to incorporate the new commands. To our knowledge this is the first time that the following wish, expressed by [Slagle 1976], is fulfilled.

> "It is an attractive idea to write a program based on mathematical logic, since this is a well-formulated and well-studied branch of mathematics. In addition, programming a computer is a way to study mathematical logic. For example, the programmer may develop powerful, natural, intuitive inference rules to which heuristics can be added easily."
>
> The Encyclopedia of Computer Science (page 1419, my emphasis).

1.1. The Research Program.

The research presented in this thesis bridges the gap between current research in the disciplines of *automatic theorem proving* and *proof checking*. Indirectly, it also relates to some research in *program verification*. Thus it is part of a collective endeavor that has a tradition of at least 20 years.

Moreover, it is a contribution to a collective effort by the Formal Reasoning group at the Stanford Artificial Intelligence Laboratory (SAIL), that represents one current of thought within the other, larger research program. This does not imply that the views expressed in this thesis are held by other members of the Formal Reasoning project or by its sponsors. It does imply, however, that this research has been guided by the author's views of this collective effort.

Nowadays, most researchers in the fields of automatic theorem proving and proof checking would agree that one of the general long term goals underlying their research is to provide practical computer systems that can be used as a research tool by working mathematicians. There are marked differences of opinion as to how this goal is to be accomplished. The purpose of this section is to give a broad overview of the main currents and their shortcomings, in order to see our contribution in its relationship to that research tradition.

AUTOMATIC THEOREM PROVING. The general goal of research in automatic theorem proving has been to produce programs that can prove mathematical theorems automatically and to find useful formalisms, decision procedures, and heuristics for this purpose. Some early researchers thought that machines would eventually surpass humans in their capacity to find proofs of mathematical theorems. While that assumption has not been disproved, progress has been generally slow and the realization of that promise does not seem to lie in the near future.

The most successful general purpose algorithm used in automatic theorem proving is the *resolution* principle by Robinson [Robinson 1965]. Many, if not the majority, of the successful theorem proving programs are based on resolution. Resolution is a *semi-decision procedure* that is *sound* and *complete* for the pure first order predicate calculus [Nilsson 1971, Luckham 1967, Lee 1967, Slagle 1971]. Thus, while it is theoretically possible to find resolution proofs of any theorem that is provable within that logical calculus, in actual practice only rather simple theorems have been proved because the size of the space of possibilities that must be searched by the computer rapidly explodes beyond the power of present days computers for more difficult theorems.

The same is true of other general purpose decision procedures. Thus much research effort is invested into finding *heuristic*[1] rules for pruning the search space. All of the more successful theorem proving programs, whether they are resolution based or not, use heuristics for guiding their search for a proof.

The problem with heuristics is that they tend to be domain specific. Just as mathematicians develop competency in particular domains of mathematics, it lies in the very nature of heuristics that they gain power by loosing generality.

Thus the effort to increase the power of theorem provers by incorporation of heuristic algorithms, inevitably leads to more specialized, domain specific theorem provers [Pastre 1978, Nevins 1975a, 1975b, Brown 1977a, 1977b, 1978, Bledsoe, Boyer and Henneman 1972, Goldstein 1973, Bundy 1973]. Given the large amount of work required to program a theorem prover, this is not a desirable state of affairs.

Because of the specialization of theorem provers, they tend to incorporate into their design the formalisms most suited for the domain for which they are intended. This forces the user to express his problems in the formalism understood by the theorem prover and thereby limits its usefulness.

PROOF CHECKING. The recognition that the correctness of proof in a logical formalism can be mechanically verified is much older than the computer. However, research into practical computer programs for this purpose came only after the initial optimism regarding the possibilities of automatic theorem provers had been tempered [McCarthy 1962, 1965, Abrahams 1963, Bledsoe and Gilbert 1967].

A proof checker is a program that incorporates the rules of a logical calculus so that it can verify that a proof is actually correct according to that calculus. For this, it needs to recognize the different objects of the calculus and to be able to perform its various inference rules.

---

[1] Heuristic: aid to discovery.

A proof checker is as general as the logical formalism it embodies. Some logical formalisms are so general that practically every domain of mathematics can be expressed in it. Thus proof checkers offer the possibility of verifying any formal proof.

Among the most general and successful proof checkers we find AUTOMATH [De Bruijn 1974], EXCHECK [Smith and Blaine 1976], and FOL [Weyhrauch 1977].

FOL is based on the well known first order predicate calculus [Mendelson 1964], and it will be discussed extensively in this thesis.

Automath is based on a new formal language developed by the leader of the Automath project, N. G. de Bruijn. The Automath language [De Bruijn 1970, 1971] is radically different from the first order predicate calculus. It was conceived as a universal language for writing mathematical books in a way that they can be proof-checked by machine, and it seems to be as powerful as first order predicate calculus but it is much less well known to the mathematical community. The largest proof checking project realized in Automath [Jutting 1977] is of a size and scope comparable to the projects undertaken in FOL, like the construction of a proof for Ramsey's theorem by this author [Weyhrauch et al. 1979]. Unlike FOL, the Automath proof checker is not interactive.

EXCHECK is an interactive proof checker for first order logic like FOL, conceived especially for mathematics instruction at the undergraduate level. It has been programmed by a group at the Institute for Mathematical Studies in the Social Sciences, and is currently used for teaching purposes at Stanford University [Suppes 1975].

Less general than FOL, but oriented by the same spirit towards *interactive construction of proofs*, are the LCF proof checkers [Milner 1972b, Gordon, Milner and Wadsworth 1977]. They are based on a formalism suited for *verification of correctness of computer programs*.

The main shortcoming of present day proof checkers is that the logical deduction steps they can check are too *atomic*, that is too small, as compared with the way humans reason. Formalizing proofs in a formalism like first order predicate calculus, or in the Automath language, is a tedious exercise comparable to programming a computer in assembly language[2]. This is the reason why [Jutting 1977, Weyhrauch et al. 1979] are probably the largest projects ever carried out on a proof checker.

In actual practice mathematicians do not attempt to produce formal proofs in a logical formalism. Their proofs are arguments whose validity is checked by other members of the mathematical community; their standards of rigor are based on a living tradition and have not been explicitly laid down. For almost all current mathematical theories, it is known that the proofs given by mathematicians can be reduced to fully formalized proofs in the first order predicate calculus [Shoenfield 1967], and some mathematicians have a fairly clear idea as to how to do this, but they would almost never bother to carry out this reduction because fully formalized proofs are very long and tedious.

In order for proof checkers to become valuable tools in mathematical practice, it will be necessary to either develop more powerful logical formalisms or to provide proof checkers

---

2 It is in fact much more difficult than assembly language programming.

with the ability to fill in many of the details of a proof. The later approach will be investigated in this thesis.

INTERACTIVE SYSTEMS. There are interactive theorem provers **[Allen and Luckham 1970, Morales 1973, Bledsoe and Bruell 1974]** and interactive proof checkers **[Weyhrauch 1977, Milner 1972b, Gordon, Milner and Wadsworth 1977]**. An interactive theorem prover attempts to remedy the limitations of theorem provers by providing the possibility of human guidance of the search for a proof. Interactive proof checkers construct the proof in an on-line conversational process with the user; this kind of system we shall call *interactive proof constructors*.

There is no clearly defined boundary between interactive theorem provers and proof constructors. The distinction rather rests on the approach that guided the development of the system, so that some systems have more of the flavor of theorem provers and others that of proof checkers. Thus an interactive theorem prover can become a tool for interactive generation of proofs **[Bledsoe and Bruell 1974]**.

On the other side, the power of an interactive proof checker can be expanded by the inclusion of theorem proving facilities; this thesis develops a methodology for this.

GOAL ORIENTED SYSTEMS. A formal proof of a theorem starts with the axioms and consists of a series of logical deductions which leads from those axioms to the theorem. Thus it has a *bottom up* structure. It is the task of mathematicians to discover new theorems they believe to be true and to prove their validity by giving proofs of them. Thus it is always the case in mathematical practice that the apparently bottom up line of reasoning of the proof has been constructed *a posteriori* to the discovery of the fact it proves, and that its construction has been guided by this fact.

Several researchers, coming from the theorem proving side, have developed *goal oriented reduction rules* to guide theorem provers towards the theorem **[Bledsoe 1971, Nevins 1974, 1976b, Ernst 1971, Brown 1977a, 1978]**. Similar reduction rules can be incorporated into an interactive proof checker. This has been done first in the earlier LCF proof checker at Stanford **[Milner 1972b]**, and then independently improved, along different lines, by the Edinburgh group **[Gordon, Milner and Wadsworth 1977]** and by us.

PROGRAM VERIFICATION. Research in program verification is related to proof checking because both problems are similar in nature. Researchers in this field look for formalisms in which the conditions of correctness of a program can be formally stated, and develop programs that can check the proofs of correctness in those formalisms. They hope that programs that verify the correctness of programs will become a practical tool in software development.

Thus one of the motivations for research in proof checking is that advances in this field are likely to serve the more practical field of program verification, in two ways: because practical computer systems for both tasks are likely to be similar, and also because the conditions of correctness of a program can be formalized in a logical language like the first order predicate calculus **[McCarthy 1963, 1966, 1977, McCarthy and Painter 1967, Cartwright and McCarthy 1979, Milner and Weyhrauch 1972a, 1972b, Weyhrauch 1975, Weyhrauch et al. 1979, Cartwright 1976, Wagner 1977]**, thus reducing one problem to the other.

INTERACTIVE PROOF CONSTRUCTION. The availability of the FOL system has spurred research in interactive construction of proofs of non-trivial theorems in various fields of mathematics [Weyhrauch et al. 1979]. Before starting work on the GOAL language described in this thesis, the author constructed a proof of Ramsey's theorem in 600 steps, and proofs of the first 98 theorems in [Kelley 1955] totalling 2000 steps. The complete proofs are presented in [Weyhrauch et al. 1979]. Because of the generality of first order predicate calculus as a means for the formalization of reasoning, the availability of FOL has also originated research into the axiomatization of several domains in this calculus [McCarthy, Sato, Hayashi and Igarashi 1978, McCarthy 1977, 1979].

To our knowledge, the only interactive proof constructor comparable to FOL is the recently developed LCF proof checker at the University of Edinburgh [Gordon, Milner and Wadsworth 1977]. Based on a formalism oriented towards program verification [Scott 1969, Scott and Strachey 1972], it is less general than FOL but it shares much of the same spirit.

We do not know of any large size proofs produced with the LCF system, but we have recently learned that they have developed a user oriented metalanguage ML for programming proof strategies [Gordon, Milner, Morris, Newey and Wadsworth 1978]. Our language has been developed independently, is quite different from theirs, and it appears to be an equally flexible tool for programming user designed strategies, except for the fact that this can be done using high level commands in ML but, for the time being, only at the LISP level in GOAL.

Because of the greater generality of FOL, theories described in LCF can be axiomatized and dealt with in FOL, while the converse is not always true. Also because of the flexibility and extensibility of GOAL, we can program in GOAL any tactics or strategies one can do in LCF. Thus, if one wishes to use FOL for some domain of knowledge for which LCF appears to be initially better suited, for instance proving assertions about recursive programs, one has first to find a suitable axiomatization in *first order logic* for that domain of knowledge [McCarthy 1977], and then one can program strategies that *simulate* the LCF deduction rules in that axiomatization. Doing so, one would have a system where there is one GOAL command for each deduction rule of LCF, and one can still chain these into more complex strategies, thereby achieving the same effects as in the LCF metalanguage.

1.2. Aims and scope of this thesis.

The research of the Formal Reasoning group at the Stanford Artificial Intelligence Laboratory is centered on the concept of interactive construction of checked proofs and is presently committed to the first order predicate calculus as an universal language for expressing mathematical reasoning. The principal computer system used by this group is an interactive proof checker for this calculus, FOL [Weyhrauch 1977, 1978a], developed and implemented mainly by Richard Weyhrauch. FOL is based on Gentzen type deduction rules [Gentzen 1935, Prawitz 1965]. In a later section, it will be described to the extent necessary for an understanding of this thesis. The research presented here depends on the availability of an interactive proof constructor. Thus we take FOL for granted and we shall not discuss the choice of the first order predicate calculus.

This doctoral thesis presents a GOAL ORIENTED COMMAND LANGUAGE, GOAL, for FOL, that has been developed and programmed by the author. To my knowledge, this is the first attempt to implement a facility of this type in an environment as general as FOL. GOAL has benefited from some ideas implemented by Weyhrauch and Milner in a goal command language for the early version of LCF [Milner 1972b, 1972a], an interactive proof checker for Scott's Logic of Computable Functions [Scott 1969, Milner 1973], that was a forerunner of FOL at the Stanford Artificial Intelligence Laboratory.

The main goal of this work has been to facilitate interactive construction of proofs by providing a facility to work in a *top down* manner, that is to work backwards from the goal (a well formed formula) towards the simpler subgoals, iterating this process until a set of formulae is obtained that can be proved more easily. When these are proved, the GOAL system produces the proof of the goal from those formulae. It does so by calling the very FOL deduction rules that, if they had been called by the user, would produce the same proof, and the proofs steps generated by GOAL are indistinguishable from those generated using the forward proving commands of FOL. We have strived to keep our system consistent with FOL in the sense just explained.

In FOL, proofs are constructed *bottom up*, that is from the simpler facts towards the goal which exists in the mind of the user. FOL offers a number of inference rules and decision procedures to carry out this task. Each inference command or decision procedure produces a new line of the proof, based on axioms and/or previous lines that must be explicitly referred to by the user.

The commands *available* in GOAL for carrying out the reduction of a goal to simpler subgoals are the inverses of FOL commands, and the GOAL commands *available for matching* (i.e., directly proving) goals use the decision procedures available in FOL.

Another aim of this work has been to provide the user of FOL with facilities for automatic generation of proofs of simple lemmas, so as to drastically reduce the amount of work necessary for interactive proof construction. This aspect takes us into the realm of automatic theorem proving, and some of the ideas are novel.

Independently, [Bledsoe 1971, Brown 1977a, Pastre 1978] have used the idea of subgoaling in theorem proving, and Bledsoe's group has developed and interactive theorem proving system. All these researchers have been concerned with theorem proving rather than proof checking.

The automatic theorem proving routines presented here are subordinated to the structure of FOL and GOAL. They operate strictly by calling the simpler reduction rules of GOAL and the decision procedures available in FOL. Thus they are heuristics for sequencing the commands available to the user, who could himself call the same sequence. It seems to be the first time that theorem proving is tackled from this angle, at least in a first order logic environment, and we understand this to be the sense of the desire shown in the quote from [Slagle 1976], in the introduction to this document.

Furthermore, GOAL has been designed so as to allow for easy addition of new reduction rules and new theorem proving facilities. These can be programmed by the user, and incorporated into GOAL by passing their names to a routine that "introduces" them to the GOAL environment, after which they can be called using the GOAL syntax.

In this way, GOAL becomes something like a programming language for automatic theorem proving. A user working on a particular domain of mathematical knowledge may observe his own behavior and identify the strategies that appear to be most fruitful in that particular domain, and may wish to program those strategies into GOAL.

The idea of a *user oriented* programming language for theorem proving applications has been developed independently by the Edinburgh group [Gordon, Milner, Morris, Newey and Wadsworth 1978], and is otherwise new. It has not been implemented at a sufficiently high level in the present version of GOAL, in the sense that the user who wishes to add new strategies will still have to understand some aspects of the GOAL code, and that for the time being these additions have to be programmed at the LISP level. But, once a certain familiarity with the code has been attained, powerful new strategies can be programmed in a few hours and simple ones in less than one hour. For future work in an interactive proof construction environment, we envision researchers having shared libraries of theorem proving strategies, documented as to the nature of applications for which they are most useful.

While it was in the initial conception of the GOAL language that it should allow for easy extension by the user, it was only after experimentation with this system that I realized the practicability of a higher level programming language for user designed strategies in a first order logic proof construction environment. In the environment of FOL and GOAL, a translator for such a language can be implemented fairly straightforwardly.

The results obtained with this approach to theorem proving are encouraging. We present here a strategy, LOGIC, that has proved a number of theorems in Set Theory, including the first 33 theorems in the Appendix in [Kelley 1955], fully automatically. More important is the fact that in most cases failure of this routine does not mean complete failure; it rather means that it carried out much of the work and it did not know how to prove one or more of the subgoals it generated. The user can then either proceed towards those unproved subgoals or cancel some branches of the goal structure that was generated and retry those goals.

Thus the GOAL language permits the FOL user to arbitrarily blend different styles of proof: the deduction oriented, bottom up style; the goal oriented, top down style; and the automatic theorem proving one.

An important building block of LOGIC is the FOL command for *syntactic simplification*. Syntactic simplification consists in recursively rewriting a formula by left to right replacements by a user specified set of equalities and equivalences. This idea is also found in Bledsoe [Bledsoe 1971] and in the LCF proof checker. It was first implemented in FOL by the author, then the code was improved by Andrew M. Robinson in order to deal with sorted variables. The FOL implementation of syntactic simplification allows for creation and naming of arbitrarily many user defined simplification sets. In GOAL some simplification sets are automatically created, used and expanded down the nodes of the goal tree. In axiomatic Set Theory, syntactic simplification turns out to be a very fruitful tactic.

The idea of syntactic simplification has already been recognized by several researchers as a powerful aid in theorem proving. In the theorem provers of [Bledsoe 1971, Pastre 1978, Brown 1977a, 1978], we find that one fixed, though perhaps extensible, set of reduction rules is presented as a *knowledge base* of the theorem prover. The knowledge bases thus presented are domain specific, often fairly large, and they substantially contribute to the power of those theorem provers.

We have found that the use of automatic simplification in theorem proving is not without problems. Sometimes it is crucial that the formulae are simplified early, at other times one wishes to postpone simplification. In FOL, one can have as many different, user designed simplification sets as one wishes, and one can add or subtract knowledge to them at any time. They can be referred to by names. In this, FOL is like the LCF proof checkers [Milner 1972b, Gordon, Milner and Wadsworth 1977].

In GOAL, the user has control over when simplification is effected, and we have strived to give him a fair amount of control over what goes into the simplification sets (or, shortly, *simpsets*) that are automatically created by the GOAL system. In any case, these automatically created *simpsets* are not used unless the user, or a strategy, requests it. In this, GOAL is unlike the goal language of the LCF proof checker, in which simplification is often done automatically, as a standarized proof mechanism, upon creation of subgoals[3].

Conditional simplification has been implemented in GOAL, in a way that is quite different from conditional simplification in the Edinburgh LCF system. In that system, conditional simplification means that the system will not simplify against certain equivalence or equality rules if there are certain variables and type variables that are shared between these rules and the hypotheses on which they depend. The details of this, as described in [Gordon, Milner and Wadsworth 1977] seem to be relevant only for an environment based on Scott's logic, but not for a first order logic environment. Also because the large amount of user control over the creation and use of simplification sets in GOAL, we have never encountered problems that would make that kind of conditional simplification necessary.

Our version of conditional simplification has been implemented only in the context of automatic theorem proving strategies, and it consists in the following: when a *WFF* is being simplified, simplification of those sub-expressions (sub-*WFFs*) that are potentially *unifiable*[4] against *VLs* in the list of facts not included in the simplification set will be inhibited.

In other words, while in the LCF system conditional simplification means that certain rules will be inhibited, we have found this unnecessary, and instead we inhibit simplification of certain parts of the wff being simplified, while leaving all of the rules active (notice that the part that is being inhibited might have been simplified not as a whole, but some part of it might have been simplified by some rule in the simplification set; our version of conditional simplification will inhibit rewriting of any subparts of the inhibited part, but the rules that could have acted on it will still be active in the rest of the *WFF*[5].

---

[3] While this is true for the early LCF proof checker developed at Stanford, the manual for Edinburgh LCF says little about the goal structure and simplification, except that "the basic outline (of simplification) remains as in the original Stanford LCF system" and that it is "the only standarized element of automatic proof in the system" (page A-39).

[4] Because they have the same structure, in the sense described in the sections on *UNIFY* in this document.

[5] See the PAIR example presented in this document.

1.3. Overview of the Goal Command Language.

GOAL consists of a tree like data structure called the *goal structure*, and of a set of commands that operate on that structure. Each node of the goal structure is a *goal*. At the top level, the user creates a goal by specifying a *WFF* to be proved and, optionally, a set of *facts* or *assertions*: axioms, previously proved lemmas, or *WFFs* to be assumed. This will become clearer in chapter 3.

From a functional point of view, there are three main types of commands: *tactics*, *matchers*, and *strategies*.

The *tactics* are commands that reduce a goal into *subgoals* (the term *goal* refers both to goals created by the user and to subgoals created by tactics). The *matchers* attempt to prove a goal directly; they either succeed or fail, but they do not attempt to reduce the goal into subgoals. The *strategies* are programmed sequences of applications of tactics and matchers. With few exceptions, the subgoals created by tactics are necessary *and* sufficient conditions for the goal to be true. Thus the goal trees are *and-trees*. We have not attempted to deal with *or-trees*, although this can be done without major modifications to the goal structure. Our reason for excluding or-trees is that they would drastically increase the search space, specially in the context of the strategies for automatic theorem proving. Where the user is controlling the expansion of the goal tree, that is by using the tactics interactively rather that using powerful search strategies, or-trees are probably an unnecessary waste of storage space.

The reduction rules incorporated in the tactics of GOAL are similar to those in [Bledsoe 1971, Brown 1977a, 1978, Pastre 1978]. These researchers used reductions of goals into subgoals as a tool in theorem proving. The most complete theoretical description of subgoaling is that of [Brown 1977a, 1978]. He views a goal as a collection of *assertions* plus a collection of *WFFs* to be proved from those assertions, and presents a set of reduction rules more complete than the other two researchers above. Almost all of these rules are present in our system, though sometimes in a different form. The main exception is his rule of skolemization on assertions, in which an existentially quantified variable of an assertion is instantiated to a Skolem function; this rule is not present in our system in all generality, and the UNIFY mechanism of FOL only partly makes up for its absence.

In order to do successful theorem proving, it is as important to operate on the facts as it is to operate on the goals. From a theoretical point of view, goals ought to be viewed as a collection of both a *WFF* and a *set of facts*, and the reduction rules ought to be described as operations on these collections, as in [Brown 1977a, 1978]. In our system, there is a mechanism of *goal preparation* that does some of the work on the facts, or assertions, of goals, and some of the tactics operate on facts. It must be admitted, however, that the treatment of assertions in GOAL lacks uniformity with respect to that of the *WFFs* of goals, and that this is a weakness from the point of view of theorem proving. On the other hand, our principal aim was to make an interactive goal command language for FOL, rather than to make a successful theorem prover. The problems encountered with the treatment of facts will be considered in more detail in the sections that deal with automatic theorem proving strategies in GOAL.

2. THE FOL SYSTEM.

2.1. Brief description.

This section gives s brief description of FOL, intended to help those readers that do not have the FOL manual [Weyhrauch 1977] at hand. A description of the more esoteric aspects of FOL, that do not concern us here, will be available shortly [Weyhrauch 1978a].

FOL is an interactive proof constructor based in the first order predicate calculus. Its deduction rules are of the Gentzen type. It has declarative commands, deduction commands, and decision procedures.

The declarative commands serve to give names to variables, constants, predicate and function symbols, and to introduce axioms. Thus various theories can be defined.

The deduction commands and the decision procedures serve to create new lines of the proof. An axiom or a line of the proof will be called a VL.[1] *VLs* have the following parts: a line number, or in the case of an axiom a name; a well formed formula (*WFF*)[2]; a list of dependencies; and a *reason* that tells how the *VL* was obtained. These parts will be explained in the sequel.

ASSUMPTIONS. A line can be assumed, using the assume command. An *assumed* line depends on itself, and any *VL* that depend on an assumed line carries with it the dependency on that assumption. Thus FOL keeps track of dependencies.

DEDUCTION RULES. Dependencies on assumptions can be *discharged* using the *deduction* command, also called *Implication Introduction*: if a *WFF* B has been proved using an assumption A, then one can deduce the *WFF* A⊃B which does not depend on A any more.

EXISTENTIAL RULES. If the main quantifier of a *VL* is the existential symbol ∃, a name can be *assumed* for the quantified variable; this is the rule of *existential specialization* or elimination. A new *VL* is generated in which the assumed name appears in place of the quantified variable. This *VL* carries a dependency on itself because of the assumed name, but this dependency cannot be discharged by the deduction command. If the assumed name disappears from (or is not free in) the *WFF* of a *VL* that has been proved with help of *VLs* that

---

[1] The word "VL" will be used extensively in this document. It can be thought of as a line of the proof, i.e. an already proved or assumed fact, if one bears in mind that axioms are to be subsumed in this concept. In FOL there are no predeclared axioms, except for the rules of the logic. Thus all axioms are entered by the user.

[2] By an *abus de langage* we will sometimes use the word *VL* to refer to the *WFF* of a *VL*. The concept of *VL* is unnecessary in mathematical logic, where a *VL* is simply a proved *WFF*, but it becomes necessary to introduce this concept when talking about the machine implementation of FOL.

depended on that assumed name, the dependency on the *VL* where the name was introduced will disappear. However, there are some exceptions to the last statement: for instance, if the same name was assumed for two different existential eliminations, and if a *VL* is generated that depends on both eliminations, then these dependencies will not disappear even when the assumed name is not present any more.

Conversely to the rule of existential specialization, there is one for *existential generalization*: any subset of the occurrences of a term in the *WFF* of a *VL* can be generalized to an existentially quantified variable.

UNIVERSAL RULES. If the main quantifier of the *WFF* of a *VL* is ∀, the quantified variable can be specialized to any term, thereby eliminating the leading quantifier. Conversely, a free variable can be generalized by introduction of the universal quantifier ∀, provided the variable is not free in any axiom or in any *VL* upon which that one is dependent.

AND/OR RULES. From two *VL*s stating A and B, respectively, a new *VL* stating A∧B can be obtained; conversely, from A∧B either A or B can be obtained. From a *VL* A and for an arbitrary *WFF* C, either A∨C or C∨A can be obtained.

REWRITE. The rewrite command effects *syntactic simplification* by a set of equivalences and/or equalities; such sets are called simpsets. Any occurrences of the left hand side of these equivalences or equalities are replaced by the corresponding right hand sides, until the process cannot be further iterated. When a *VL* is given to the rewrite command, an equivalent *VL* is produced and added to the proof. When a *WFF* is given, if it rewrites to TRUE this *WFF* is added to the proof as a new *VL*; if it rewrites to a different equivalent *WFF*, a new *VL* stating this equivalence is generated. When a term is given and it rewrites to a syntactically different term, the equality of the two expressions is stated in a new *VL*.

Simpsets are defined by specifying a set of axioms and/or *VL*s. When new *VL*s are obtained by the rewrite command, the simpset is part of the *reason* of the new *VL*, which depends on any *VL*s of the simpset *that were actually used in the simplification process*. That is, the rewrite command is smart enough so it does not make the new *VL* depend on the dependencies of all the *VL*s in the simpset, but only on those that were applied as rewrite rules in that particular call to the command. Rewrite, simplification sets, and *match trees*[3] are explained in pages 49 through 55 of the FOL manual [Weyhrauch 1977]. The rewrite command obeys the following syntax.

---

3 Simplification sets are represented internally by LISP objects called *match trees*. But a user can think of the two words: *simpset* and *match tree*, as synonyms. What is important, from the user's point of view, is that sets of rewrite rules can be stored and referred to by a *identifiers*. These identifiers must be declared to be of type *simpset*.

Syntax:

REWRITE ALT[<WFF> | <VL>] BY <simpsetexpr> ;

Simplification set expressions are defined by the syntax below, where "," means to take the union of the given expressions. The binding powers of "," , "∪" and "\" are that "," binds least strongly, "\" has an intermediate binding power, and "∪" is strongest.

Syntax:

<simpsetexpr> := { <vllist> } | <simpset> |
<simpsetexpr> , <simpsetexpr> |
<simpsetexpr> ∪ <simpsetexpr> |
<simpsetexpr> \ <simpsetexpr>

In this BNF form, a *simpset* is an identifier that has been declared to be of type *simpset*. And a *VLLIST* is a list of *VL*s and axioms, separated by commas. To form a *simpsetexpr*, that list must be enclosed in curly brackets: {}.

DECISION PROCEDURES. FOL has several decision procedures. One of these is TAUT. If a *WFF* is a tautology, or if it follows tautologically from a set of axioms and *VL*s, a new *VL* stating this *WFF* can be obtained by the TAUT command. The new *VL* depends on the union of the dependencies of the *VL*s that the user said were necessary to obtain the new one. That is, this command is not as smart as *REWRITE* in eliminating unnecessary dependencies; for instance, if the *WFF* is a ground tautology *per se* but the user said it follows tautologically from a certain *VL* that has dependencies, these will be carried over.

Similar to *TAUT* is TAUTEQ, that includes the rules of equality. Other decision procedures are: MONADIC, that decides validity of *WFF*s whose *prenex normal form* [Mendelson 1964] is such that all universal quantifiers precede all existential ones[4]. UNIFY, a decision procedure that matches quantified *WFF*s whose *matrices*[5] are *isomorphic*[6] and attempts to find a set of solutions to the quantified variables. *UNIFY* was developed by R. Weyhrauch and A. Chandra, and is as yet undocumented[7].

Sometimes *REWRITE* acts as a proof procedure: namely when the *WFF* rewrites to *TRUE*, in which case the *WFF* is stated as a new *VL*. The same happens with SIMPLIFY, a command for *semantic simplification* that will not be discussed in this thesis.

---

[4] *If this seems confusing, see also the footnote about MONADIC in the section on matchers in the next chapter.*

[5] The *WFF* that remains after removal of the leading quantifiers.

[6] In the sense that they have the same structure of logical connectives.

[7] UNIFY is *not* related to the *unification* algorithm that is used in resolution theorem proving. May be it ought to be renamed to avoid this confusion.

RESOLVE. A variation of *UNIFY* is *RESOLVE*. If a *VL* is a disjunction, perhaps preceded by some quantifiers, and the negation of one of the disjuncts can be unified against another *VL*, the other disjunct can be stated as a new *VL*, where some of the quantified variables are instantiated according to the solutions obtained from the unification of the other. At present RESOLVE has some bugs and some unresolved theoretical problems, nevertheless it has been used in the GOAL because it is a powerful command for the purposes of automatic proof generation.

SORTS. In FOL, variables can be declared to be of some *sort*. Predicates and functions can be declared to take arguments of some sort. Functions can be declared to produce terms of some sort. Thus some terms are recognized by FOL as being of a certain sort. Some sorts can be declared to be at least as general as others using the MOREGENERAL declarative command. For instance, in several versions of Set Theory there are sets and classes, the later being more general than the former.

Sorts affect many of the previously mentioned commands. In particular, they affect the quantifier rules and the simplification commands. They also affect the UNIFY command, but the current version of UNIFY does not take sorts into account.

Sorts introduce many complications, some of which have not yet found a satisfactory solution. They shall not be dealt with in this thesis.

ADMINISTRATIVE COMMANDS. There are also some strictly administrative commands, the most important one being the SHOW command, used to display axioms, *VLs*, declarations, and proofs. In GOAL there is an analog to the show command. Another important one is the CANCEL command, used to erase a proof or an arbitrary *end segment* of it; that is, all the *VLs* with line numbers greater than or equal to the number passed as argument. There is also a GOAL analog to this command.

2.2. The style of proof construction in FOL.

FOL has no facilities other than GOAL for goal oriented proof construction. Formal proofs in FOL are much longer than the informal proofs of mathematics; this is true even for the more formal domains like axiomatic Set Theory. The user has to type at least as many commands as there are *VLs* in the proof.

When constructing a proof, it is often difficult to keep track of its overall structure because one's attention tends to get caught in the detail. This is because the commands are so *atomic*: facts that appear obvious to the mathematician often require a dozen or more commands and a considerable amount of detail work.

This problem does not rest with FOL, but with the first order predicate calculus. Logicians seldom use this calculus to prove any theorems; rather, they study it in order to make sure that their theorems *can* be proved in the calculus. When they expound formal theories in

books, the majority of the proofs given do not fill in all the details. These proofs aim at convincing that one knows how to fill in the missing details. Complete formal proofs of some simple theorems are only given as pedagogic examples. However, no formalisms that are convincingly more powerful and equally general as this calculus are known at present.

One way of aleviating this problem is to add to FOL facilities for automatic generation of proofs of "obvious" facts. Another is to look for commands that produce shorter proofs. Of the later kind, the simplification commands are very useful; so are also the decision procedures TAUT, TAUTEQ, MONADIC and UNIFY and the related RESOLVE command. Of the first kind are the *strategies* for automatic proof generation described in this thesis.

Yet the principal way in which GOAL attempts to aleviate the problem is by providing a facility for *goal oriented*, *top down* proof construction. In any case, the final proof looks the same; but the tree-like goal structure can be used as a recordkeeping facility that remembers the structure of the proof and can be referenced at any time when the user wishes to remind himself of what remains to be done.

3. THE STRUCTURE OF GOAL.

3.1. Overview.

This chapter describes GOAL. First it describes the data structure upon which GOAL commands operate, called the *goal structure*. Then it describes the GOAL commands.

The data structure is a list of goal trees. Each goal tree is a recursive data structure in which all nodes have the same structure. The root of the tree is a *top level goal*, any nodes below are *subgoals*. The term *goal* refers to either. Top level goals are created by the user using the *GOAL* command. Any other goals are created by the *tactics* described below.

There are several types of GOAL commands. The GOAL command that creates top level goals. The ABANDON command that prunes a branch of a goal tree. There is also an administrative *showgoal* command. But the most important GOAL command is *TRY*. It is used to invoke the *operative elements* that operate on the goal structure. There are three types of operative elements: *tactics*, *matchers*, and *strategies*.

The *tactics* create new subgoals by decomposing a goal. The *matchers* attempt to prove a *bottom level* goal, or *leaf* of a goal tree, directly. The *strategies* are programmed sequences of applications of tactics and matchers.

Goals have *statuses*; the three mutually exclusive statuses are: *untried*, *tried*, and *proved*. At any time, the leaves of a goal tree are either untried or proved, and the other nodes are tried. *Trying* a goal means invoking an operative element on it. Only untried goals can be tried. However, trying a goal changes its status only if the operative element succeeds; then it becomes either proved or tried. Tried (but not proved) goals can be *abandoned*, in which case they become again untried.

The difference between the three types of operative elements can be defined precisely with regard to the GOAL code. However, from the point of view of the functional characteristics of the operative elements, this classification is not as clear cut: some tactics may succeed in proving a goal directly, in which case they act like a matcher; and some strategies may do little more than a tactic, while others may be powerful theorem provers.

Each goal has a number of *parts*, some of which may be empty. These parts carry data that is used and changed in various ways by the GOAL commands that operate on and change the goal structure. Among the parts of a goal we find *facts* and *simpsets*. The operation of *trying* a goal has a *side effect* called *preparation* of the goal, that often introduces changes to these parts. The special command *prepare* can be invoked by the user to provoke this side effect without actually trying the goal; this may add new facts or simpsets to the goal and new lines to the proof.

Goals can be referred to by a numbering system. In most GOAL commands, the user can

either give an explicit reference to a goal or use the default for that command. There are three basic defaults: the *current*, the *next*, and the *last* goal. These are pointed to by global variables that change dynamically as the man-machine conversation unfolds.

3.2. Goals.

The Goal Structure is roughly speaking the converse of the proof structure in FOL. In the proof structure, new lines of the proof are produced by invoking FOL inference commands or decision procedures. In the goal structure, the user specifies at the top level the *WFF* to be proved, giving also some information as to the facts that need be used and how they will be used. *Tactics* decompose this *WFF* into sets of subgoals. The subgoals are sufficient, and with a few exceptions also necessary, conditions for the original goal to be true.

This process of tearing apart goals can be applied recursively so that a tree structure is generated. At any moment, the leaves of the tree represent sufficient conditions for the root of the tree to be true, and the system knows how to produce a proof of the original goal when all the leaves have been proved.

Top level goals are those created by the user directly. Invocation of tactics create subgoals of a goal, which we call its sons. Thus, top level goals are those that do not have a parent. The sons of a goal behave in every respect like a goal, therefore the term *goal* will refer indistinctly to goals at any level in the tree.

At any time, a goal has one of the following *statuses*:

UNTRIED: it has no sons and it has not been proved;

TRIED: it has sons (these have been necessarily created by a *tactic*);

PROVED: the *WFF* of the goal has become a line of the proof (and the structure remembers the number of that line).

When the last son of a *tried* goal is proved, the system immediately *proves* that goal; that is to say, it applies some deduction rule of FOL to the lines that correspond to the proved sons, thereby generating a new line of the proof that matches the *WFF* of the goal whose status then becomes *proved*. We call this process *unwinding*; its result is a FOL proof that looks the same as one generated by a user of FOL.

When a goal is *proved*, its sons are removed and cannot be accessed any more (i.e., they will be eventually disposed of by the LISP garbage collector).

3.3. Treatment of assertions (or facts).

This section offers an overview of the treatment of *facts* or assertions in GOAL. It refers to several concepts that will be explained in detail in the following sections. Facts are treated mainly by the *prepare* mechanism. A complete description of this subject will be given in the section on goal preparation.

It has been said in the introduction, that one should view goals as sets of *WFFs* to be proved and sets of *facts* or assertions. In the implementation of GOAL, the facts are attached to the goal as an **a-list**. The facts are axioms or *VLs*. The user can also specify *WFFs* to be attached to this list; in this case, the *preparation* mechanism (that will be explained later) *assumes* these *WFFs* using the FOL command *assume*; thus they become *VLs*.

The facts of a goal are passed down to its sons. Often new facts are added to sons. Thus, with a few exceptions, the facts of a goal are a subset of the facts of its sons.

The user can specify facts in two ways: using *assume* or *sassume*. The second option causes the fact to be included into the list of *simplification rules* (simpsets) attached to the goal.

Besides those facts given by the user, we find facts created by the mechanisms of GOAL. Some tactics create new facts: for instance, when an implicational *WFF* of the form A⊃B is tried by the "⊃I" tactic, a goal B is obtained and A is assumed (or sassumed, depending on the structure of A). Also, when a goal is proved but some of its *brothers* are still unproved, that goal is added to the facts of those unproved *brothers* and of their *descendants* as well. There are still other ways in which new facts are generated; these will be discussed when we explain the *prepare* mechanism.

GOAL does not offer the user as much control over the *facts* as it does with respect to the treatment of the *WFF* of the goal. This can be seen as a drawback because it limits the kinds of strategies that can be easily programmed.

It should be mentioned that there are two parts of a goal that hold facts: they are called *FACTS* and *ADDEDFACTS*. Facts added to a goal, either upon its creation or later, usually go to *ADDEDFACTS*, except for those created by the *prepare* mechanism itself. This mechanism empties *ADDEDFACTS* and passes its contents over to *FACTS*. There are several reasons of implementation why we chose to do things that way; one of the effects obtained is that *WFFs* given by the user using *ASSUME* and *SASSUME* are not added to the proof or put into the simpsets until the goal is actually tried; the same delayed effect applies for other transformations the *prepare* mechanism does to the facts.

3.4. The parts of a goal.

The following parts are imbedded in the structure of unproved goals. When the goal is *untried*, many of these parts are NIL. *Proved* goals have a different structure: they just have a goal number, a *VL* (as opposed to a *WFF*), and sometimes a *reason*[1] that indicates how they were obtained.

GOAL NUMBERS. They number *brother* goals starting with 1. Brother goals are those that have a common parent; also the top level goals are considered to be brothers. Thus goals can be referred by means of a list of natural numbers, each one preceeded by the token "#". For instance: **#3#1#1#2** means *the second son of the first son of the first son of the third element of the list of top level goals*

GOALWFF. The *WFF* of the goal.

DESCENDANTS. The list of sons; these are goals.

REASON. Indicates how its sons were obtained; it contains all the necessary information so the unwinding mechanism can prove the *WFF* of the goal by referring to the *VL*s that prove its sons.

FACTS. A collection of pointers to *VL*s that are stored with the goal; these *VL*s are used by the matchers in various ways when trying to prove the goal; they are also used sometimes by "CASES". They are stored in a list of association lists, because they may be used in a number of different ways. Some of them may be assumptions indicated by the user, or created by the GOAL system, or proved sub-goals that are *brothers* of the goal or of some of *ancestor* of it.

SIMPSETLIST. A list of simpsets associated with a goal. It would be more logical to condense all these simpsets into just one. That simpset would have to be expanded and shrunk dynamically when the goal tree is created and traversed, and this poses problems of implementation that make it more convenient to store lists of simpsets instead.

SIMPSETREASONLIST. A list of the *VL*s and names of simpsets in the *SIMPSETLIST*, so that the system can produce *reasons* for the steps of the proof it generates, in the same way FOL does. (Reasons for proof steps indicate how the *VL*s are obtained in FOL).

SIMPSETADDFLAG. A flag indicating whether additions have occurred to the *SIMPSETLIST*;

---

[1] This is not to be confused with the *reason* of a *VL* nor with the *REASON* of a tried goal.

this flag is used by some automatic theorem proving strategies in order to know whether it makes sense to attempt rewriting anew.

ADDEDFACTS. Any information contained here is eventually passed over to *FACTS*; here there may be *VLs* or *WFFs*, indicated by the user or produced by the system; it was thought convenient to have a separate list of this kind, because it permits to treat *FACTS* more uniformly and also because it indicates whether any new facts have been added to the goal since the last time it was tried (this information is used by some automatic theorem proving strategies).

QUANTELIMLIST. A list of the quantifier eliminations made down the goal tree; this has many uses; it keeps track of bindings made in *brother* branches of the tree, to assumed existential eliminations in the proof, so as to know whether a match may be such that the proof would not unwind. It is also used by UNIFY so as to reconstruct some matches that could not otherwise be unified. In these ways GOAL makes a limited amount of *skolemization*.

3.5. Skolemization and the Quantelimlist.

To *Skolemize* an existentially quantified variable in a goal is to eliminate the quantifier and to replace the quantified variable by a variable name that matches any term of the same sort. An analogous operation can be done on an universally quantified variable in an assertion of the goal [Brown 1977a, 1978].

For example: if a goal is $\forall x.\exists y.\forall z.P(x,y,z)$, and we do an universal, an existential, and an universal subgoaling operation, we obtain as a goal: $P(x,y,z)$. But x, y, and z ought to have a different status in that subgoal: x and z have to be *free* variables, while y could be matched against (almost) any term. More precisely, y can be matched against *any term that does not depend on* z; for instance, against a term $t(x)$ which contains some free occurrences of x. *Skolemizing* in this case means subgoaling to: $P(x,f(x),z)$, where $f(x)$ is a *Skolem function* of the variable x. The use of *Skolem functions* in theorem proving is discussed in a number of textbooks, for instance in [Nilsson 1971].

In GOAL, Skolemization is performed by recording quantifier eliminations in the *QUANTELIMLIST*. When a variable that has been Skolemized in this way is matched at some node in the goal tree, then the same variable cannot be matched again to a different term at some other node, i.e. it is not *free* any more. The *QUANTELIMLIST* keeps track of such bindings and records the node where the binding was made. The abandon command sometimes *frees* again a variable that has been bound in this way: namely, it does so when the node at which the binding was performed lies below the goal being abandoned.

For example, if the original goal were $\forall x.\exists y.\forall z.(P(x,y,z)\wedge Q(x,y,z))$, after several subgoaling operations we may have the two subgoals $P(x,y,z)$ and $Q(x,y,z)$. In this case GOAL would remember, for either one of this two subgoals, the series of universal and existential subgoaling operations that were performed down the goal tree. It would be able to match the

variable *y*, in either subgoal, against an arbitrary term provided it does not contain any free occurrences of *z*. Now suppose that one of the subgoals is matched. Say *Q(x,y,z)* is matched against *Q(x,t(x),z)*, for some term *t(x)*. After that, GOAL will refuse to match *y* in *P(x,y,z)* against anything else but *t(x)*. We say *y* has become *bound* to *t(x)*.

Now, what if the choice of *t(x)* was wrong in the first place, so that the user wants to take it back? Both subgoals have a common parent, which is *P(x,y,z)∧Q(x,y,z)*. Upon this parent (or some ancestor of it) being *abandoned*[2], GOAL will free *y* so that it can again be matched with some other appropriate term.

Further illustration of the use of this feature of GOAL can be found later in this manuscript: in the *PAIR example* shown in the section on automatic theorem proving, and in the description of the matcher *EQUNIFY*.

3.6. Unwinding.

When a sub-goal (i.e. any goal that has a parent) is proved either by a matcher or by the unwinding mechanism, its parent is looked at. If all the sons of that parent are *proved*, the proof of the parent is produced; otherwise, the just proved sub-goal is added to the *ADDEDFACTS* of its unproved *brothers* (the unproved sons of its parent), and of the *descendants* of these, so they will be used by the matchers and sometimes added to the simpsets (depending on the structure of the *WFF* of the proved goal).

When a goal is matched, the unwinding mechanism also looks at the *QUANTELIMLIST* and checks whether a *Skolemized* variable in the *GOALWFF* has been matched in a way that makes it depend on some existential elimination in the proof. I.e., it checks whether any variables that came from existential eliminations performed in the goal tree appear as assumed names for existential eliminations in any of the *VLs* on which the newly proved goal depends. If this is the case, the said variables are *bound* in the *QUANTELIMLIST*, and these bindings carry over to all the nodes that descend from the node where that existential elimination was performed. For a proper understanding of this, the reader is referred to the documentation of the FOL existential elimination rule in [Weyhrauch 1977].

3.7. Defaults: current, next and last goal.

There are three defaults. They are kept track by global variables. Initially they are all *NIL*. If the user defaults by not specifying an optional argument in a call to a command and the default variable for that command is *NIL* at that time, the ensuing error message indicates that the command does not know what goal to try. The defaults obey to the following rules.

---

[2] With the *ABANDON* or with the *RETRY* command.

NEXTGOAL. It is an untried goal. This is the default for the TRY command. Thus it can be thought of as *the next goal to be tried*. It is the last goal *created*, *prepared*, or *abandoned*, either by the user or by some GOAL command.

LASTGOAL. The last goal decomposed by an invocation of the *TRY* command. That is, successfully tried by a user invoked tactic, or tried by a strategy that succeeds in decomposing it.

CURRENTGOAL. The last goal tried by any tactic or matcher.

The *unwinding* mechanism causes the following irregularities in the rules above: it resets *CURRENTGOAL* to the father of the last goal proved by either a matcher or the unwinding mechanism, and *NEXTGOAL* to some unproved son of *CURRENTGOAL*, that is to a brother of the last proved goal. If *LASTGOAL* becomes proved, then it is reset to the same as *CURRENTGOAL*. When a top level goal is proved, all three defaults become *NIL*.

3.8. The GOAL commands.

3.8.1. Goal creation.

GOAL. This command is used to create a top level goal. The user must specify the *WFF* and can also indicate assumptions, sassumptions and simpsets. A *sassumption* is an assumption that gets also added to the simpset. The assumptions can be *WFFs*, *VLs*, or axioms. Those that are *WFFs* are written onto the proof by the *ASSUME* command of FOL when the goal is tried. By default, the special simpsets LOGICTREE, and COMPTREE (automatic instantiation of the axiom scheme of comprehension for sets), are included, but the user can prevent this by saying "NOTREES".

Syntax:

```
GOAL <WFF> [OPT ASSUME REPT(ALT[ <WFF> | <VL> ])]
           [OPT SASSUME REPT(ALT[ <WFF> | <VL> ])]
           [OPT SIMPSET <simpsetexpr> ]
           [OPT NOTREES ]  ;
```

Along with the syntax of GOAL commands, we shall show examples of their use. In this first example we start with some FOL definitions in order to set up the context of our examples.

Let us recall that the five asterisks: "*****" is the prompting response of FOL. Most user's commands end with one semicolon, except the *AXIOM* command that end with a double semicolon. What comes after the semicolon, up to the next "*****", is the response of GOAL or FOL.

Example:

*****DECLARE INDVAR x y z z1 u v w;

*****DECLARE PREDCONST ∈ 2 [INF];

*****AXIOM EXTENT: ∀x y.(x=y≡∀u.(u∈x≡u∈y));;

EXTENT: ∀x y.(x=y≡∀u.(u∈x≡u∈y))

*****AXIOM PAIR: ∀x y.∃w.∀u.(u∈w≡(u=x∨u=y));;

PAIR: ∀x y.∃w.∀u.(u∈w≡(u=x∨u=y))

*****GOAL ∀x y.∃z.(∀w.(w∈z≡(w=x∨w=y))∧∀z1.(∀w.(w∈z1≡(w=x∨w=y))⊃z1=z))
ASSUME PAIR SASSUME EXTENT;

Goal #1: ∀x y.∃z.(∀w.(w∈z≡(w=x∨w=y))∧∀z1.(∀w.(w∈z1≡(w=x∨w=y))⊃z1=z))

*****

3.8.2. Referencing goals.

Many commands take a goal as argument. Goals are referenced by a numbering system. A **goal reference** is a list of natural numbers, each preceded by the token **#**. The first one is the number of a top level goal; the next is the number of one of its sons; the following, the number of a son of that son; and so forth. An error message ensues when a nonexistent goal is referenced.

Syntax:

<goalref> := REPT(# <natnum>)

Examples:

#3#1#1#2

#1

3.8.3. Addition of Facts to a Goal.

ADDFACTS. This command is used to add facts to an already existing, untried goal. It uses almost the same syntax as the *GOAL* command, except that a goal reference must appear instead of the *WFF*. This command does not have any default; the goal reference must be explicit.

Syntax:

```
ADDFACTS <goalref> [OPT ASSUME REPT(ALT[ <WFF> | <VL> ])]
                   [OPT SASSUME REPT(ALT[ <WFF> | <VL> ])]
                   [OPT SIMPSET <simpsetexpr>]
                   [OPT NOTREES]  ;
```

The following two commands achieve the same effect as the previous example of goal creation.

Example:

```
*****GOAL ∀x y.∃z.(∀w.(w∈z≡(w=x∨w=y))∧∀z1.(∀w.(w∈z1≡(w=x∨w=y))⊃z1=z));
Goal #1:  ∀x y.∃z.(∀w.(w∈z≡(w=x∨w=y))∧∀z1.(∀w.(w∈z1≡(w=x∨w=y))⊃z1=z))
*****ADDFACTS #1 ASSUME PAIR SASSUME EXTENT;
*****
```

3.8.4. Trying Goals.

The operative elements of GOAL are the tactics, matchers, and strategies. All of these are called by the TRY command, using the same initial syntax; however, many of these require additional information, that is parsed by the *parser* associated with that operative element. This additional information is given at the end of the TRY command; its syntax depends on the particular operative element and will be described in the sections on tactics, strategies, and matchers.

TRY. This command is used to apply a tactic, strategy, or matcher, to a goal. The user may specify the goal in two different ways: by a goal reference; or by an natural number, meaning the number of a son of CURRENTGOAL. Otherwise the default NEXTGOAL is tried (if this is NIL, an error message is given). The user has to specify the tactic, strategy, or matcher, and give any additional information that may be required by that particular operative element. Only an *untried* goal may be tried by this command. In the syntax below, *op_name* is the name of a tactic, matcher, or strategy; and *op_info* is the additional information required by that operative element (possibly none); these two items will be described, for each element, in the section on the operative elements.

Syntax:

TRY [OPT ALT[<goalref> | natnum]] USING <op_name> <op_info> ;

where

"<op_name>" is the name of a tactic, matcher, or strategy, and

"<op_info>" is any additional, specific information required by that element.

Only *untried* goals may be tried; a goal whose status is *tried* can be *abandoned* and then tried again. The following command combines these two functions.

RETRY. Combines ABANDON and TRY. This command does not admit a default: it requires an explicit goal reference. If the goal is *untried*, it will be accepted and tried.

Syntax:

RETRY <goalref> USING <op_name> <op_info> ;

For an illustration of the use of this command, see the examples in the section on the matcher *UNIFY*.

3.8.6. QED.

The QED command is to be used *only* when the *GOALWFF* is exactly equal (except for the names of bound variables) to that of a *VL*. It does not cause any new line to be added to the proof, instead it records that the goal is proved by that *VL* and it invokes the unwinding mechanism.

The two arguments are optional. The defaults are: NEXTGOAL for the goal reference, and the last *VL* in the proof, for the *VL*.

Syntax:

QED [OPT <goalref>] [OPT <VL>];

3.8.6. Abandoning Goals.

ABANDON. Applied to a *tried* goal, it makes it *untried* by garbage collecting its sons. The user may specify the goal number, or by default the last goal that was tried is abandoned.

Syntax:

ABANDON [OPT <goalref>] ;

For an illustration of the use of this command, see the examples in the section on the matcher *UNIFY*.

3.8.7. User invoked preparation.

PREPARE. This command invokes the preparation mechanism without actually *trying* a goal; its main use is for causing the assumed *WFFs* of a goal to be written onto the proof. It has a "PLUS" switch that can be used to add facts to the goal. It uses the same initial syntax as TRY for referring the goal, and it has the same default, NEXTGOAL. It does *not* reset any of the defaults.

Syntax:

PREPARE [OPT ALT[<goalref> | <natnum>]] ;

3.8.8. Displaying goals.

SHOWGOAL. It displays the goals together with their attached properties. It is a very verbose command, but it has a *TERSE* option. If no arguments are given, all top level goals and all of their descendants are displayed. Optional arguments are: 1) a goal reference, or one of

the words: *nextgoal*, *lastgoal*, or *currentgoal*; in this case only that goal and its descendants are displayed; 2) "DEPTH <natural number>", in which case descendants only down to a certain level are displayed (the number can be 0); 3) "TERSE" for the terse option. The terse option is recommended for seeing the goal tree in perspective. The verbose option is useful for examining the parts of a goal; in this case it is recommended to use a small *depth*, 1 or 0, in order to limit the size of the typed response.

Syntax:

```
SHOWGOAL [OPT ALT[ <goalref> | NEXTGOAL | LASTGOAL | CURRENTGOAL ] ]
         [OPT DEPTH  <integer> ]
         [OPT TERSE]  ;
```

Examples:

```
*****SHOWGOAL;

Goal #1:  ∀x y.∃z.(∀w.(w∈z≡(w=x∨w=y))∧∀z1.(∀w.(w∈z1≡(w=x∨w=y))⊃z1=z))
    VLSASSU:  EXTENT  ∀x y.(x=y≡∀u.(u∈x≡u∈y))
    VLASSU:  PAIR  ∀x y.∃w.∀u.(u∈w≡(u=x∨u=y))
      Simpsets:  ( BY  LOGICTREE  COMPTREE)

*****SHOWGOAL TERSE;

Goal #1:  ∀x y.∃z.(∀w.(w∈z≡(w=x∨w=y))∧∀z1.(∀w.(w∈z1≡(w=x∨w=y))⊃z1=z))

*****
```

3.9. The operative elements of GOAL.

The building blocks of GOAL are its *operative elements*: the *tactics*, the *strategies*, and the *matchers*. GOAL has been designed to allow for easy addition of new operative elements; in the section on expanding GOAL, we shall look at the structure of the operative elements in more detail. For this section, it is enough to know that each operative element has a *name* and a *parsing routine* associated with it.

All the operative elements are called by the *TRY* command. As we described that command, we introduced two syntactic items: *op_name* and *op_info*. In this section, we shall look at the operative elements that are now present in GOAL. For each one, its function will be described, and the two syntactic items above will be defined.

3.9.1. Tactics.

The tactics attempt to decompose the goal into (expectedly simpler) subgoals. Most tactics transform the *GOALWFF*, the main exception being the tactic *CASES*. Any successful application of a tactic produces one or more sons of the goal. Tactics do not try to decompose those sons any further. The status of the goal becomes *tried*; the status of the newly created sons is *untried*.

Most tactics create subgoals that are *necessary* and *sufficient* conditions for the goal to be true; but some create subgoals that are only *sufficient*; when the later is the case, we shall state it explicitly as we describe the tactic.

At present we have the following tactics.

3.9.1.1. Universal rule: ∀I.

The main symbol of the *GOALWFF* must be "∀". The *matrix* of the *WFF* is produced as a subgoal, i.e., the leading universals are eliminated. By default, the quantified variables are instantiated to the same variable names, but a different instantiation can be specified by the user. The optional op_info is a list of *variable names* without repetitions; the parser also checks whether these variables are free in some axiom and whether they are of sort at least as general as the quantified variable, and gives error messages if it finds conditions that would make it impossible to unwind the proof. The standard name *∀I* refers to the FOL rule by which the proof of the goal will be produced in the unwinding process.

<op_name> := ∀I | UG | ug

<op_info> := OPT[REPT[<variable name>]]

The following examples start with the goal created in the previous example of the *GOAL* command. They form a sequence of commands, except where noted otherwise. Thus the default **nextgoal**, which is the last goal created, applies to most of them.

Example:

*****TRY USING ∀I;

Goal #1#1: ∃z.(∀w.(w∈z≡(w=x∨w=y))∧∀z1.(∀w.(w∈z1≡(w=x∨w=y))⊃z1=z))

*****

3.9.1.2. Existential rule: ∃I.

The main symbol of the *GOALWFF* must be "∃". The *matrix* of the *WFF* is produced as a subgoal, i.e., the leading existentials are eliminated. By default, the quantified variables are instantiated to the same variable names, but a different instantiation can be specified by the user. The optional op_info is a list of *terms*. If these terms already appear in the *WFF*, a list of occurrences is kept so the proof will unwind properly.

<op_name> := ∃I | EG | eg

<op_info> := OPT[REPT[<term>]]

Example:

*****TRY USING ∃I;

Goal *1*1*1: ∀w.(w∈z≡(w=x∨w=y))∧∀z1.(∀w.(w∈z1≡(w=x∨w=y))⊃z1=z)

*****

3.9.1.3. Conjunction rule: ∧I.

The main symbol of the *GOALWFF* must be "∧". The two conjuncts are produced as subgoals. Op_info is nil.

<op_name> := ∧I | AI | ai

Example:

*****TRY USING ∧I;

Goal *1*1*1*1: ∀w.(w∈z≡(w=x∨w=y))
Goal *1*1*1*2: ∀z1.(∀w.(w∈z1≡(w=x∨w=y))⊃z1=z)

*****

3.9.1.4. Equivalence rule: ≡I.

The main symbol of the *GOALWFF* must be "≡". Two subgoals are produced: if the *GOALWFF* is " A ≡ B ", the subgoals are " A ⊃ B " and " B ⊃ A ". Op_info is nil.

<op_name> := ≡I | EQUIV | equiv

Example:

*****TRY 1 USING ∀I;

Goal *1*1*1*1*1: w∈z≡(w=x∨w=y)

*****TRY USING ≡I;

Goal *1*1*1*1*1*1: w∈z⊃(w=x∨w=y)
Goal *1*1*1*1*1*2: (w=x∨w=y)⊃w∈z

*****

3.9.1.5. Deduction rule: ⊃I.

The main symbol of the *GOALWFF* must be "⊃". One subgoal is produced: if the *GOALWFF* is "A ⊃ B", the wff of the subgoal is " B ", and " A " is added to it as an *assumption* or *sassumption*. Whether it will be a *sassumption* or not, i.e. whether it will be added to the simpset, depends on a test performed by the preparation mechanism; it will if it is an equivalence or equality, possibly preceded by some universal quantifications. Op_info is nil.

<op_name> := ⊃I | DED | ded

Example:

*****TRY USING ⊃I;

Goal *1*1*1*1*1*2*1: w∈z

*****

In the last example, the antecedent w=xvw=y has been attached to the goal as a *WFF* to be assumed. When the goal is *tried*, or *prepared*, that *WFF* will be written unto the FOL proof. The next example will show this.

3.9.1.6. Rule of CASES.

The basic idea behind this tactic can be expressed in the following tautology:

$$((A \vee B) \supset C) \equiv (A \supset C) \wedge (B \supset C).$$

But the tactic can be used in several ways, depending on the arguments given by the user in the optional *op_info*.

If the argument is an axiom or *VL*, this must be a disjunction, possibly preceded by some existential quantifications; then, if the *GOALWFF* is, say, " C ", and the axiom or *VL* is, say, " A ∨ B ", the following two subgoals are produced: " A ⊃ C " and " B ⊃ C "; if that axiom or *VL* is already among the *facts* of the goal being tried, it is removed from the *facts* of the sons.

If no argument is given, the tactic *searches for a disjunction*, possibly preceded by existential quantifications, among the *facts* of the goal, and proceeds as above.

The argument can also be a *WFF*, say, "A"; this produces cases on the tautology "A ∨ ¬A".

<op_name> := CASES | cases

<op_info> := OPT[VL | WFF]

Example:

```
*****TRY USING CASES;

1 w=xvw=y (1)

Goal *1*1*1*1*1*1*2*1*1: w=x⊃w∈z
Goal *1*1*1*1*1*1*2*1*2: w=y⊃w∈z

*****
```

3.9.1.7. Syntactic simplification: REWRITE.

The *GOALWFF* is *re-written* using the syntactic simplifier. The user may specify a *simpset-expression*; in addition to it, all the simpsets attached to the *SIMPSETLIST* of the goal are used.

If the *GOALWFF* rewrites to TRUE, this tactic acts like a matcher; if it rewrites to a *WFF* other than the original one, one subgoal is produced, and the *SIMPSETREASONLIST* attached to the goal, plus the user specified simpset-expression, are stored in the *REASON* of the goal.

When this tactic is called from some strategy, one can use a special flag to inhibit simplification against *WFFs* in the *FACTS* of the goal that have the same structure[3] as the expression or sub-expression being matched, because such sub-expressions are potentially *unifiable* against those *facts* at a lower level in the goal tree. This option will be explained in greater detail in the section on the LOGIC strategy.

<op_name> := REWRITE | rewrite

<op_info> := OPT[ALT[BY | by] <simpsetexpr>]

The syntax of <simpsetexpr> has been given in the previous chapter on FOL.

Example:

*****TRY USING REWRITE BY {EXTENT};

Goal #1#1#1#1#1#2#1#2#1: ∀u.(u∈w≡u∈y)⊃w∈z

*****

Since the axiom EXTENT was attached to the goal by the *SASSUME* option when the goal was created, the shorter version "TRY USING REWRITE" would have the same effect in the last example.

3.9.1.8. Semantic simplification: SIMPLIFY.

The *GOALWFF* is *simplified* by the semantic simplifier using any *semantic attachments* that are current. (The FOL mechanism of semantic attachment will be described in a forthcoming publication by Weyhrauch.)

---

[3] in the sense of the UNIFY command. See UNIFY.

If the *GOALWFF* simplifies to TRUE, this tactic acts like a matcher; if it simplifies to a *WFF* other than the original one, one subgoal is produced.

An example of the use of this tactic is the following: if the *GOALWFF* contains some subexpression *SET(x)*, where x is a variable of *sort SET*, or of some *less general sort*, that subexpression will simplify to *TRUE* and the original *WFF* will simplify too.

At present, <op_info> is nil.

<op_name> := SIMPLIFY | simplify

3.9.1.9. Special tactics.

We have at present three other tactics. The first two, **IMPLICATION** and ∨I (*or*-Introduction), constitute an exception to the rule that the subgoals are not only *sufficient*, but also *necessary* conditions for the goal. The third one, **INDUCTION**, is special purpose: it was designed for the work on *Ramsey's theorem* and it assumes and that the name of the empty set is the *individual constant* λ.

3.9.1.9.1. Disjunction rule: ∨I.

This tactic is used to subgoal to *one* of the disjuncts of a *GOALWFF* whose main quantifier is "∨". It produces only one subgoal to the goal. The user has to specify "1" or "2", meaning the first or second conjunct is to become the *GOALWFF* of the subgoal.

<op_name> := ∨I | ORI | ori

<op_info> := 1 | 2

Example:

*****TRY *1*1*1*1*1*1 USING ⊃I;

Goal *1*1*1*1*1*1*1: w=x∨w=y

*****TRY USING ∨I 1;

2 w<z (2)

Goal *1*1*1*1*1*1*1*1: w=x

*****

3.9.1.9.2. Implication rule.

Often times there is an implicational *WFF*, or perhaps a universally quantified implication among the *facts* of the goal, such that it is possible to prove its *antecedent* and that the *consequent* would make it easy to prove the goal. Or there may be a *VL* in the proof that has those properties.

The op_info for this tactic is optional. The user may specify a *VL* whose *WFF* is a (possibly universally quantified) implication. If it is universally quantified, a list of instantiations for the universally quantified variables may be given.

If no op_info is given, the tactic attempts to find a *VL* with the required characteristics among the *facts* of the goal. If it finds, it will still try to find some instantiation for the leading universal quantifiers that would cause the *GOALWFF* to *match* against the consequent; the tactic will fail if this does not succeed. The reason for making this tactic so "careful" is because of its intended use in automatic theorem proving strategies: in those, we are concerned with avoiding an explosion of the search space.

If the tactic succeeds, the *antecedent* of the implication becomes the *GOALWFF* of the subgoal. When the goal has been proved, the unwinding mechanism will first prove the *consequent* by calling the FOL command *RESOLVE* on the following two *VL*s: the just proved *antecedent* and the *fact* from which this *antecedent* was extracted. After this, the unwinder will attempt to match the goal against the *VL* that proves the *consequent*.

<op_name> := IMPLICATION | implication

<op_info> := OPT[<VL> OPT[REP[<variable name>]]]

3.9.1.9.3. Induction rule.

This tactic was designed for our work on Ramsey's theorem. It is assumed that the empty set is the individual constant λ^4. It checks that the *GOALWFF* is universally quantified and that the variable bound by the first quantifier is of *sort NATNUM*.

It creates two subgoals: if the *GOALWFF* is $\forall i.PRED(I)$, then the subgoals are $PRED(\lambda)$ and $\forall i.(PRED(i) \supset PRED(SUC(i)))$, where SUC is assumed to be the name of the *successor* function.

<op_name> := INDUCTION | induction

---

4 The reasons for this choice are only historical. The user wishing to use 0 instead can change this tactic by redefining its components, as will be explained in the next section.

3.9.2. Matchers.

The matchers attempt to prove the *GOALWFF* directly, that is without decomposing the goal, by using some decision procedures of FOL and the *facts* of the goal. We have four matchers at present, the main one being *UNIFY*. Their functions correspond to the FOL commands by the same names. We have an additional special purpose matcher that does not exist as a FOL command, *EQUNIFY*, in order to deal with a special case that *UNIFY* cannot handle.

UNIFY and *EQUNIFY* inspect the *QUANTELIMLIST* of the goal and use it to reconstruct some possible quantifier introductions, from those eliminations recorded in that list. *Skolemization* is achieved to a limited extent in this way.

3.9.2.1. UNIFY.

This is the main matcher. It uses the undocumented FOL procedure *UNIFY* written by Weyhrauch and Chandra. This procedure attempts to match a *WFF* against an already proved one, if both *WFF*s have the same structure of logical connectives after removal of the leading quantifiers. The FOL command is further documented in [Weyhrauch 1977], and the algorithm will be documented in a forthcoming paper by Weyhrauch.

If the user specifies a *VL*, the matcher attempts unification only against this one; otherwise it does so against each one of the *VL*s in the *facts* of the goal.

<op_name> := UNIFY | unify

<op_info> := OPT[<VL>]

The matcher does more than the FOL command: for each one of the *VL*s against which it attempts unification, it loops trying to reconstruct the existential quantifier eliminations that were made previously in the goal tree.

For instance, assume that we are unifying against a *VL* that says ∀x.∃y.P(x,y). The FOL command will unify ∃y.P(z,y) but not P(x,y) against it, and this is indeed correct. However, if we have a subgoal P(z,w) and w is recorded in the *QUANTELIMLIST* as coming from some application of the tactic ∃I and being still *free*, then this matcher will produce unification against the *VL* above.

Example:

```
*****ABANDON *1*1*1*1;

Goal *1*1*1*1: ∀w.(w<z=(w=x∨w=y))     abandoned.
```

*****TRY USING UNIFY PAIR;

3 ∃z.∀w.(w∈z≡(w=x∨w=y))

4 ∀w.(w∈z≡(w=x∨w=y)) (4)

*****

In the last example, it is unnecessary to specify "PAIR" in the call to *UNIFY*, because this axiom is already in the list of *assumed* facts of the goal. Also using *RETRY* would make it unnecessary to *abandon* the goal. The following equivalent example shows a shorter way of obtaining the same effect.

Example:

*****RETRY *1*1*1*1 USING UNIFY;

Goal *1*1*1*1: ∀w.(w∈z≡(w=x∨w=y)) abandoned.

3 ∃z.∀w.(w∈z≡(w=x∨w=y))

4 ∀w.(w∈z≡(w=x∨w=y)) (4)

*****

3.9.2.1.1. EQUNIFY.

This is a special purpose matcher designed to deal with the following special case that UNIFY does not handle. Suppose the goal is "x=y" and y is free in the *QUANTELIMLIST*; that is, this subgoal was part of a goal "∃y.(x=y∧P(x,y))". Then the matcher UNIFY will fail on this equality; the user can match it by calling EQUNIFY.

<op_name> := EQUNIFY | equnify

Now we shall start up a new example in order to show the use of this matcher. The following dialog shows also the effects of two matching attempts that failed because of user's error. See also the explanation after the example.

Example:

*****DECLARE PREDCONST P 2;

*****DECLARE INDCONST λ;

*****AXIOM REFL: ∀x.P(x,x);;

REFL: ∀x.P(x,x)

*****GOAL ∃x.(x=λ∧P(x,λ));

Goal *2: ∃x.(x=λ∧P(x,λ))

*****TRY USING ∃I;

Goal *2*1: x=λ∧P(x,λ)

*****TRY USING ∧I;

Goal *2*1*1: x=λ
Goal *2*1*2: P(x,λ)

*****TRY USING EQUNIFY;

The wff of this goal is not an equality.

*****TRY 1 USING EQUNIFY;

3 ∃x.x=λ

4 x=λ (4)

*****TRY USING UNIFY REFL;

No unification.
The tactic UNIFY can't be applied to goal
Goal *2*1*2: P(x,λ)

*****TRY USING REWRITE;

Goal *2*1*2*1: P(λ,λ)

*****TRY USING UNIFY REFL;

5 P(λ,λ)

6 P(x,λ)≡P(λ,λ) (4)

7 P(x,λ) (4)

8 x=λ∧P(x,λ) (4)

9 ∃x.(x=λ∧P(x,λ))

*****

In the first call to *EQUNIFY*, the default nextgoal was goal #2#1#2, whereas the user wanted to apply the matcher to goal #2#1#1; the next time he does this correctly. *EQUNIFY* recognizes the fact that the variable *x* in the goal *x*=λ can be matched against any term, so it matches it with *x* itself.

As the next call to *UNIFY* fails, the user recognizes that he must first *rewrite* the goal. He could have said: "TRY USING REWRITE 4;", that is stating explicitly that the fact that *x*=λ must be used to *rewrite* the goal. But this was unnecessary because the goal structure will automatically use a fact proved in one branch of the goal tree in order to *fertilize* the *sibling branch*.

After the last call to UNIFY, we can see the FOL proof being produced by the *unwinding* mechanism.

3.9.2.2. TAUT and TAUTEQ.

These two matchers use the FOL commands by the same names. They take any number of *VL*s as optional arguments. They attempt to prove that the *GOALWFF* follows tautologically from the collection of *facts* attached to the goal plus the *VL-list* specified by the user.

The FOL command TAUT decides ground tautologies, while TAUTEQ adds the rules of equality. One should bear in mind that TAUTEQ is much slower that TAUT.

Using the op_name TAUTO the user can call both matchers at the same time. In this case, TAUT is invoked first and the TAUTEQ is invoked if TAUT failed.

<op_name> := TAUT | taut | TAUTEQ | tauteq | TAUTO | tauto

<op_info> := OPT[<VL-list>]

We shall rehearse the last example once more in order to show the use of TAUTEQ.

Example:

*****TRY USING ∧I;

Goal #2#1#1: x=λ

```
Goal *2*1*2: P(x,λ)
*****TRY 1 USING EQUNIFY;
3 ∃x.x=λ
4 x=λ  (4)
*****VE REFL λ;
5 P(λ,λ)
*****TRY USING TAUTEQ ↑;
6 P(x,λ)  (4)
7 x=λ∧P(x,λ)  (4)
8 ∃x.(x=λ∧P(x,λ))
*****
```

The command "VE REFL λ", after using EQUNIFY, is ~ FOL command. The call to the matcher TAUTEQ indicates that the last line of the proof must be used; of course it is also necessary to use line 4, but GOAL will do that in any case.

3.9.2.3. MONADIC.

This matchers uses the FOL command by the same name. Its syntax looks the same as that of TAUT and TAUTEQ, but, unlike these, it does not attempt to match against the whole collection of *facts* attached to the goal. There are two sets of reasons for this difference; we shall discuss them below in this section.

If the user does not specify a *VL-list*, the matcher attempts to prove that the *WFF* is *TRUE* by itself. Otherwise it tries to prove that it follows, by the *MONADIC* decision procedure, from the conjunction of those *VL*s.

<op_name> := MONADIC | monadic

<op_info> := OPT[<VL-list>]

The FOL decision procedure MONADIC was programmed by Bill Glassmire. Its name refers to the *monadic predicate calculus*. The pure monadic predicate calculus is known to be a *decidable theory* [Mendelson 1964]. However, since FOL deals with theories other than the monadic predicate calculus, the actual implementation of this command makes it into a decision

Procedure for *WFF*s in *universal-existential prenex normal form*[6].

Thus, if the *WFF* being decided does not reduce to that form, MONADIC recognizes it falls out of its scope and informs the user accordingly. This is the first reason why we do not wish to attach the whole list of *facts* of the goal to the *VL-list* given by the user. For it is likely there will be *WFF*s, among the facts, that do not reduce to universal-existential prenex form.

Fortunately, if A and B reduce to that form, so does A∧B. Thus it would be theoretically possible to keep track of which facts do reduce to it, and always add those facts to the *VL*s given by the user when calling the *MONADIC* matcher. Doing so would greatly enhance the power of this matcher, as well as the power of automatic theorem proving strategies like LOGIC. This could be done easily if we were not running up against the physical limitations of our machine. MONADIC uses an enormous amount of computing resources, and it often causes LISP to run out of free storage. Thus we found that, if the list of those *facts* that do reduce to the desired form is passed to *MONADIC* by default, the automatic theorem proving strategies tend to abort most of the time for that reason.

Now let us rehearse the last example one more time.

Example:

*****GOAL ∃x.(x=λ∧P(x,λ));

Goal #2: ∃x.(x=λ∧P(x,λ))

*****TRY USING MONADIC;

The MONADIC command decided that this formula is not valid.

*****TRY USING MONADIC REFL;

3 ∃x.(x=λ∧P(x,λ))

*****

---

[6] Bill Glassmire's implementation of MONADIC has not been documented and I am not familiar with it. Richard Weyhrauch offered the following commentary: "MONADIC was implemented by Bill using Quine's method of reducing monadic sentences to sentences of the form ∀∀∀∀∃∃∃∃ called variously "universal-existential", "AE", or "∀∃". The decision procedure for these was well known in the thirties. MONADIC actually uses the more general decision procedure to decide ∀∃ formula that it has found. Function symbols are handled in some reasonably but ad hoc way. I am not sure how." (Personal communication).

3.9.3. Strategies.

The strategies are called using the same syntax as the tactics and matchers, by the TRY command. They effect calls to tactics and/or matchers. They may be very complex, or quite simple. From the point of view of the GOAL code, any routine that after decomposing a goal may attempt to either decompose, or match, any of the subgoals it created, is to be classified as an strategy. The reason for this is that calls to tactics and matchers are mediated by one *master routine which can be applied only to untried goals, thus being impossible to mediate* calls to entities that call tactics through the same master routine.

At present we have three strategies, only one of which is a theorem prover. It is very easy to add others. We have not done so because one of our aims was to develop one powerful theorem proving strategy within the context of FOL and GOAL.

3.9.3.1. LOGIC.

This is our automatic theorem prover. As it will be described in detail in a special section, here we shall only present its syntax. The optional op_info field begins with the word PLUS and serves to add new elements to the *FACTS* of the goal; when using this switch, the user does not have control over the *assume/sassume* option; instead, the *prepare* mechanism will decide which *VL*s go into the simpset in the same manner it decides for *VL*s that are generated by the goal structure.

<op_name> := LOGIC | logic

<op_info> := OPT[PLUS <VL-list>]

When called on a subgoal (i.e., on a goal that is not a top level one), if LOGIC succeeds in proving it, it will backup further in the goal tree, attempting to prove all of its *relatives*: i.e., any unproved descendants of any one of its ancestors.

3.9.3.2. ELIMINATION.

This strategy does not attempt to prove anything, i.e. it does not call any matchers. It tries to recursively decompose the *WFF* using the following tactics: $\forall$I, $\exists$I, $\wedge$I, $\equiv$I, $\supset$I, and CASES.

There can be no conflict of priorities between the first five tactics above, for each one of the can be applied only to *WFF*s whose main quantifier is the one indicated by the name of the tactic. However, there may be a conflict with cases, for both CASES and one of those tactics can be applied to the same goal. This conflict is always resolved to the disadvantage of CASES.

ELIMINATION does not call the tactics until it has checked that they can be applied, that is by looking up the leading quantifier of the *WFF*, or by calling a routine that checks for the existence of a disjunctive *assertion* in the *factlist*.

The optional op_info serves to limit the *depth* of the recursion. If used, elimination proceeds at most to the maximum depth indicated, down the tree, starting from the goal. Otherwise it decomposes it as far down as possible.

<op_name> := ELIMINATION | elimination

<op_info> := OPT[<DEPTH> <natnum>]

The following examples are self-explanatory; again we are rehearsing some of the previous examples.

Examples:

```
*****ABANDON #1;

Goal #1: ∀x y.∃z.(∀w.(w∈z≡(w=x∨w=y))∧∀z1.(∀w.(w∈z1≡(w=x∨w=y))⊃z1=z))%
    abandoned.

*****TRY USING ELIMINATION;

Goal #1#1: ∃z.(∀w.(w∈z≡(w=x∨w=y))∧∀z1.(∀w.(w∈z1≡(w=x∨w=y))⊃z1=z))
Goal #1#1#1: ∀w.(w∈z≡(w=x∨w=y))∧∀z1.(∀w.(w∈z1≡(w=x∨w=y))⊃z1=z)
Goal #1#1#1#1: ∀w.(w∈z≡(w=x∨w=y))
Goal #1#1#1#2: ∀z1.(∀w.(w∈z1≡(w=x∨w=y))⊃z1=z)
Goal #1#1#1#1#1: w∈z≡(w=x∨w=y)
Goal #1#1#1#1#1#1: w∈z⊃(w=x∨w=y)
Goal #1#1#1#1#1#2: (w=x∨w=y)⊃w∈z
Goal #1#1#1#1#1#1#1: w=x∨w=y
Goal #1#1#1#1#1#2#1: w∈z
3 w=x∨w=y  (3)

Goal #1#1#1#1#1#2#1#1: w=x⊃w∈z
Goal #1#1#1#1#1#2#1#2: w=y⊃w∈z
Goal #1#1#1#1#1#2#1#1#1: w∈z
Goal #1#1#1#1#1#2#1#2#1: w∈z
Goal #1#1#1#2#1: ∀w.(w∈z1≡(w=x∨w=y))⊃z1=z
Goal #1#1#1#2#1#1: z1=z

*****RETRY USING ELIMINATION DEPTH 4;

RETRY USING ELIMINATION DEPTH 4;
     ↑ ∧
A goal number reference  is required here.

*****RETRY #1 USING ELIMINATION DEPTH 4;

Goal #1: ∀x y.∃z.(∀w.(w∈z≡(w=x∨w=y))∧∀z1.(∀w.(w∈z1≡(w=x∨w=y))⊃z1=z))%
```

abandoned.

```
Goal *1*1: ∃z.(∀w.(w∈z≡(w=x∨w=y))∧∀z1.(∀w.(w∈z1≡(w=x∨w=y))⊃z1=z))
Goal *1*1*1: ∀w.(w∈z≡(w=x∨w=y))∧∀z1.(∀w.(w∈z1≡(w=x∨w=y))⊃z1=z)
Goal *1*1*1*1: ∀w.(w∈z≡(w=x∨w=y))
Goal *1*1*1*2: ∀z1.(∀w.(w∈z1≡(w=x∨w=y))⊃z1=z)
Goal *1*1*1*1*1: w∈z≡(w=x∨w=y)
Goal *1*1*1*2*1: ∀w.(w∈z1≡(w=x∨w=y))⊃z1=z

*****
```

3.9.3.3. IFCASES.

This is a special purpose strategy for *conditional expressions*. Conditional expressions **[McCarthy 1963]** are legal in FOL: there are both *IF-WFFs* and *IF-terms*. There are two special simpsets for conditional expressions: WFFIFTREE and ARGIFTREE. The first deals *IF-WFFs*, the second deals with *IF-WFFs* and *IF-terms*.

In the present implementation, the user must specify a *WFF* as argument to IFCASES. First the strategy calls *CASES* on this *WFF* and ⊃I on both subgoals; then it calls the tactic REWRITE on both *grandsons* of the goal, making sure to include in the simpset: *WFFIFTREE*, *ARGIFTREE*, and the *antecedent* of the just effected ⊃I, in each case (that is, the *WFF* given by the user and its negation, respectively).

<op_name> := IFCASES | ifcases

<op_info> := < WFF >

A variation of this strategy has been used in the example of the Takeuchi function, that is presented in a separate chapter.

4. EXTENDING GOAL.

While the previous chapter described GOAL from the user's point of view, the purpose of this chapter is to introduce the reader to the programming of new operative elements. At present this cannot be done without considering the FOL code. However, the documentation in the following sections should be very helpful to any one wishing to extend GOAL. We shall look at some internal aspects of the GOAL implementation; in particular, at the system that controls the activity of the operative elements.

It is always difficult to present a total system in a linear manner, and even more difficult for the reader to find his way through the maze. Necessarily, this is only a partial description; a user will still have to look at the code when trying to program extensions to GOAL. We shall follow an unconventional approach, trying to present the material in a sequence intended to make it easy to read. Thus we shall circle several times over some aspects, gaining depth each time. We shall begin with some general information about the GOAL implementation.

The strategies are easier to program than the other two types of operative elements, and they are also, expectedly, the most frequent and useful type of extension that users will want to make. Strategies are easier than matchers and tactics because the latter interact more with the FOL routines; hence more knowledge of the FOL code is required to program these. Strategies are almost entirely contained in GOAL; they are not concerned with unwinding nor with updating the *goal structure*. But they perform nevertheless some operations that require some knowledge of the FOL implementation: for example, a knowledge of the internal representation of *wffs* in FOL is needed in order to determine which is the leading quantifier of a *wff*.

Parts of goals can be accessed using MLISP macros that bear the same name as those parts. Parts of *WFFs*, and *VLs*, are accessed using MLISP macros defined in the FOL code. Readers desiring to do their own strategies should look at the MLISP code of the existing operative elements in order to get acquainted with these macros.

4.1. The three components of the operative elements.

With each *tactic* there are three associated routines: the *parser*, the *executer*, and the *unwinder*. The other two types of operative elements do not need an *unwinder*, but do have a *parser* and an *executer*.

The *executer* performs the required actions: in the case of tactics, it creates subgoals; for matchers, it calls the FOL decision procedures; and in the case of strategies, it calls other operative elements. The *parser* parses user's calls (by the *TRY* command) to the operative element. And the *unwinder*, that is automatically called when all sons of a goal have been proved, produces the FOL forward proof of that goal, from the *VLs* that prove its sons.

In order to program a new operative element, the user has to supply the *executer*, the

parser, and the *unwinder* if the element is a *tactic*. It is also necessary to provide a name for the atom that will represent the new element, and to call a routine that *introduces* this element to the system; there is one such routine for each type of operative element.

4.2. The internal representation of the operative elements.

The components of an operative element, that is the routines mentioned in the previous section, are stored in the *property list* of a LISP *atom* that represents the operative element.

The name of this atom will be referred to as the *standard name*[1] of the operative element.

The names of operative elements are stored in the global variable *OPELEMLIST*. The global variable *STRATEGYLIST* is a subset of *OPELEMLIST*. The routines that *introduce* new operative elements will refuse to introduce an element whose standard name is already in this first list. However, they will *not* check whether the names of the associated routines provided by the user conflict with other identifier names used in the system. It is the user's responsibility to make sure that no names are duplicated.

4.3. The control system.

In this section we shall cover the structure of the subsystem of GOAL that controls the activity of the operative elements. This system is the core of GOAL, as well as its only extensible part. It is entered by the *TRY* command.

The three routines associated with each operative element do *not* communicate with each other directly. They are managed by *master routines* that control the operations of: parsing, execution, and unwinding. These master routines are: *TRY* which controls parsing; *TRYING* which controls execution of both tactics and matchers, *TRYCMPL* which controls execution of strategies, and *UNWIND*, that is called either from *TRYING* or recursively by itself, and which controls unwinding.

The only one of these master routines which must be called directly by the user is *TRYING*: this is the case in user programmed strategies. The others are mentioned because, in order to program new operative elements, it is helpful to have a general understanding of the control structure. *TRYING* will be dealt with in a special section. The basic conventions to be observed will be described in the sections that explain how to program the different types of operative elements.

The only operative elements that can call other operative elements, or themselves recursively, are strategies. However, when they call a tactic or a matcher, the call must

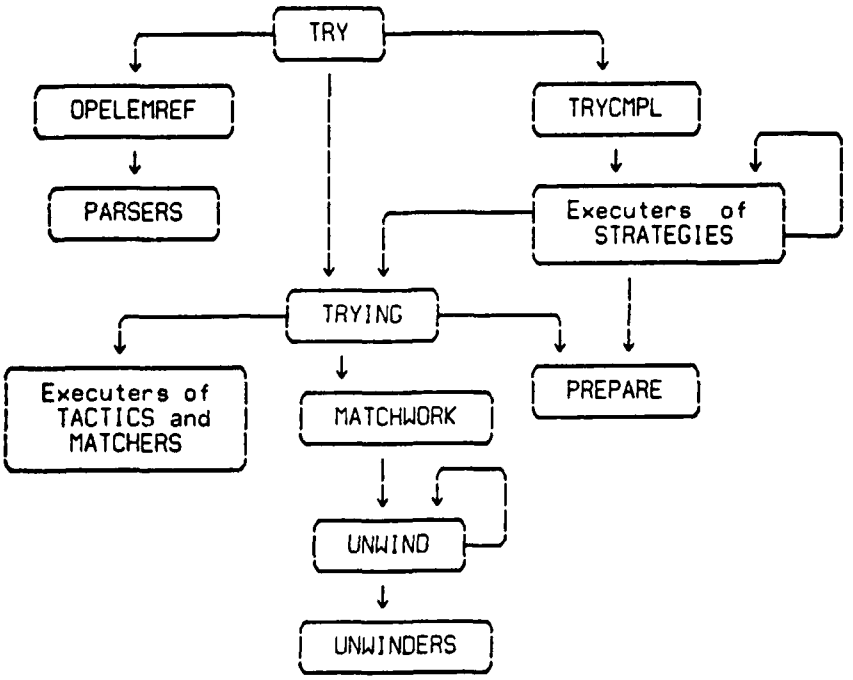
---

[1] It is a standard name from the point of view of the system, but it does not need to be the same name with which the tactic is invoked by the user via the TRY command.

always be routed as a call to *TRYING*. We shall later see in more detail how to do this. Calling any tactic or matcher directly, without mediating the call through *TRYING*, will always result in a fatal error. On the other side, a strategy calling another strategy can, and should, make the call directly to the executer; thus the executer of a strategy can recursively, directly call itself. Strategies can also call *PREPARE*.

The hierarchical structure of this system is shown in figure 1. The arrows indicate *possible* calls from one routine to another; they do not indicate calls that will always occur. Possible recursions are indicated accordingly.

4.3.1. FIGURE 1: Structure of TRY.



4.4. Types of variables.

The reader must now become aware of the important distinction between the two following types of variables that are used by the operative routines of GOAL: *goals* and *threads*.

A variable of type *goal* is a pointer to a *goal*; *threads* are described in the next section.

The goal structure is generally updated using the LISP functions *RPLACA* and *RPLACD*. Thus any local variables of type *thread* or *goal* will undergo the same updates.

4.4.1. Threads.

Most of the time, the GOAL routines are operating on some goal. However, they often need to be able to find its parent, or to detect whether it is a top level goal. Sometimes it is also necessary to determine whether a goal is an *ancestor* of another.

For these reasons we have chosen *threads* as the most common way of pointing to goals. Many routines pass *threads* to each other as arguments, but some take just a *goal* as argument.

The *thread* associated with a goal is a *list* whose car is the goal, and whose cdr is the *thread* of its parent. Thus the *goal of a thread* is the car of the thread. The last element of a thread is always the global variable *GOALLIST*, which is the list of *top level goals*.

4.4.2. The three defaults.

The global variables that identify the three *defaults* discussed in chapter 3 are called: *LASTGOALTHREAD*, *NEXTGOALTHREAD*, and *CURRENTGOALTHREAD*.

The user should never assign values to these three variables. They are automatically reset by the system. However, users may want to use local variables to keep track of threads in a strategy[2].

A thread is *empty* if it is a list of only one element, namely *GOALLIST*. The *macro*[3] *EMPTYTHREAD(THREAD)* checks whether a thread is empty. The cdr of the thread of a *top level goal* is empty.

The routine *SUBTHREAD(THR1,THR2)* checks whether the goal of THR1 is an *ancestor* of the goal of THR2. This is equivalent to THR1 being equal to an *end segment* of the list THR2.

---

2 For instance, the strategy LOGIC uses a queue of threads in order to implement a breadth first search.

3 Here the word *macro* refers to a MLISP macro. A number of macros have been used to name the different parts of goals, and for some other purposes. They are expanded when the MLISP code is translated into UCI-LISP.

4.4.3. Status checking.

A *proved* goal has a structure totally different from that of an *unproved* one. Trying to access parts of a *proved* goal as if it were *unproved* will result in fatal errors. Also, a *tried* goal cannot be tried again by any operative element unless it has previously been abandoned.

Thus, when programming *strategies* it may be necessary to check the status of a goal. There are the following status checking predicates: the *MLISP macros PROVED* and *UNTRIED*, and the function *TRIED*. All of these take only one argument, of type *goal*.

4.4.3.1. Abandoning.

Abandoning goals can be done from within strategies using the function *ABNDN(THR,PSWT)*. The first argument is a *thread*. The second a *printswitch*: if this switch is NIL, then no message will be printed when the goal is abandoned.

4.5. Rules for programming new operative elements.

Now we shall outline the conventions for programming the different components of operative elements. This description cannot be exhaustive.

We shall begin with the easiest, namely the *parsers*.

In each case, we shall end the section with an example.

4.5.1. Parsers.

Parsers take only one argument of type *goal*. The rules for the returned expression will be described below.

Let us recall that the syntax of the *TRY* command is:

TRY [OPT ALT[<goalref> | natnum]] USING <op_name> <op_info> ;

The parsers parse *<op_name>* and *<op_info>*. The rest of the above syntax *including* the semicolon is *not* parsed by the parser. Thus it is most important that the higher level parsing routines expect a semicolon, after the parser returns control.

The syntax for the *<op_name>* and the *<op_info>* is defined by the user in the act of

programming the parser. The *<op_name>* is usually an alternative of two words (i.e., upper or lower case). The *<op_info>* may be more involved; for instance, in the case of the *REWRITE* tactic, the <op_info> may recognize a FOL expression for a *simpset* and call the FOL routine *SIMPSETEXPR* that constructs the internal representation of a simpset.

4.6.1.1. The expression returned by parsers.

1) If the parser does *not* recognize the *<op_name>*, then it must return NIL.

2) If the parser recognizes the *<op_name>*, it proceeds to parse the *<op_info>*.

3) If the scanned expression does not conform to the syntax for the *<op_info>*, the system must pop up to the top level of FOL, while normally issuing some error message. There are various ways of doing this, which will be illustrated in the examples; the FOL routine ENDM is very useful for poping up.

4) If it is detected at parse time that the operative element cannot be applied to the goal, then return a LISP *atom*. This atom will be considered to describe the name of the element and will be printed in a message by the *TRY* command.

5) **Successful parsing**: a *list* must be returned; the first element must be the *standard name* of the operative element (i.e. the atom that represents this element internally). The following elements of the list are going to be the additional arguments taken by the executer, if any; this point requires some further explanation.

We shall see that the first two arguments of any executer are: the *thread* of the goal, and a *printswitch*. Some executers take additional arguments; these additional arguments are to be passed in the list returned by the parser, and must be in the same order.

Thus, if the executer takes only the two standard arguments, the parser must return the standard name *consed* with NIL. If the executer takes, say, four arguments, the parser must return a list of three elements; the second and third elements of this list will be the third and fourth arguments taken by the executer.

4.6.1.2. Examples of parsers.

We shall now look at the MLISP code of several parsers and comment them.

4.6.1.2.1. Conjunction rule: ∧I.

```
EXPR PARSAND(G);
      IF CHECKI('AI,'ai,'?∧) THEN IF MAINSYM(goalwff(G))='?∧ THEN <'?∧I> ELSE '?∧I  ;
```

The FOL routine CHECKI checks for the occurrence of the token: "AI" or "aI" or "∧I". The FOL MLISP macro MAINSYM returns the leading connective of a *WFF*. The GOAL MLISP macro goalwff returns the *goalwff* of a goal. The standard name of this tactic is the quoted atom "∧I".

Thus this parser returns: NIL if the *<op_name>* is not recognized, the quoted atom "∧I" if the tactic cannot be applied to the goal, and the *standard name* "∧I" *consed* with NIL if it can. The *<op_info>* for this tactic is nil.

In the second case, the TRY command will issue the following message:

"The tactic ∧I can't be applied to goal ... "

and then it will display the goal.

Notice that there is no check for a semicolon in this parser; the command is expected to end here, and the check for the ending semicolon is performed at a higher level.

4.6.1.2.2. Disjunction rule: ∨I.

```
EXPR PARSOR(G:SP);
      IF CHECKI('ORI,'ori,'?∨) ∧ (SP←NATNUM#()) THEN
            IF MAINSYM(goalwff(G))='?∨ THEN
                  IF (SP=1) ∨ SP=2 THEN <'?∨I,SP>
                  ELSE PARSORMSG()
            ELSE '?∨I  ;
```

SP is here a local variable to hold the <op_info>, which must be 1 or 2, depending on which one of the two disjuncts will become the new subgoal. NATNUM# is a FOL routine that *expects* a natural number and pops up to the top level, while issuing an error message, if anything else is encountered.

Thus this parser first checks that the *<op_info>* is a natural number. If it is not, it will pop up to the top level and the error message will be the standard FOL message that indicates the type of token expected, with an arrow pointing to it. However, if a natural number is encountered, this parser will perform a second check to determine that it is 1 or 2;

this check comes *after* examining the leading quantifier of the *goalwff*; admittedly, the order of this checks could be somewhat different.

In case of successful parsing, the returned expression is a list of two elements. The second element, SP, will be passed as the third argument to the executer of this tactic, which takes exactly three arguments.

PARSORMSG is a routine that prints an error message, specifically for this parser. The user can add such routines to enhance the quality of error messages in the parsers. We can learn something from the code of PARSORMSG.

```
EXPR PARSORMSG();
      BEGIN TERPRI();
      PRINC("The argument to PARSOR must be 1 or 2.");
      ENDM();
      END;
```

We see that PARSORMSG does a carriage return, prints a message, and then it calls the FOL routine ENDM.

ENDM is a FOL routine that ends scanning of a command line and pops up to the top level of FOL.

4.6.1.2.3. The rule of CASES.

This is a more complicated parser.

```
EXPR PARSECASES(G);
      BEGIN NEW X;
      IF TK2@('CASES,'cases) THEN
            IF TK='?; THEN IF X←EXISTORASSU(G,'?v ) THEN RETURN(<'CASES ,X>)
                        ELSE CASEPARSEMSG3(" disjunction",'CASES)
            ELSE IF X←WFF⊛(NIL) THEN RETURN(<'CASES ,X>)
            ELSE IF X←VL⊛⊛(NIL) THEN IF MAINCONN(WFFOF(X),'?v ,NIL,T)
                                          THEN RETURN(<'CASES ,X>)
                                       ELSE CASEPARSEMSG2(CAR X,'?v)
            ELSE CASEPARSEMSG1('CASES);
       RETURN(NIL);
       END;
```

The FOL routine TK2@ is used to parse an alternative of two tokens.

The global variable TK also belongs to FOL; at any time during command scanning, it

contains the *next* token in the input stream; thus the condition " IF TK='?; " checks whether the next token is a semicolon but it does *not* perform scanning of this token; this is important, because scanning would advance the scanner, and we know that the parser must stop short of the command closing semicolon.

This check for the semicolon is done because the *<op_info>* for this tactic is optional. If no *<op_info>* is given, the parser calls EXISTORASSU to determine whether among the *facts* of this goal there is some disjunction. EXISTORASSU is rather involved and will not be presented here.

WFF⊛ and VL*⊛ are FOL parsing routines that recognize *WFFs* and *VLs*, respectively.

4.5.1.2.4. The tautology matcher.

We also show the code of the parser that combines the *TAUT* and *TAUTEQ* rules of FOL, because the code of the corresponding executer will be shown in a later section.

```
EXPR PARSETAUT(G);
  IF TK2@('TAUT , 'taut ) THEN RETURN( < 'TAUT, 3, VLLIST⊛(NIL) ,NIL> )
  ELSE IF TK2@('TAUTEQ , 'tauteq ) THEN RETURN( < 'TAUT , 4, VLLIST⊛(NIL) ,NIL> )
  ELSE IF TK2@('TAUTO , 'tauto ) THEN RETURN( < 'TAUT , 5, VLLIST⊛(NIL) ,NIL> );
```

4.5.1.2.5. The elimination strategy.

The following is another example of an interface between a parser and an executer. The executer of this strategy will be shown later.

```
EXPR PARSELIM(G:DEPTH); IF TK2@('ELIMINATION,'elimination) THEN
  <'ELIM , IF TK2@('DEPTH,'depth) THEN
              IF DEPTH←NATNUM@() THEN DEPTH ELSE ENDM() ELSE 1000 > ;
```

4.5.2. Executers in general.

The first argument of any executer is the *thread* of the goal, and the second argument is a *printswitch*. Additional arguments are optional. If the *printswitch* is NIL, printing of the generated subgoals would be inhibited.

The executers for the three types of elements perform different functions and will be described separately. However, the executers of *tactics* and *matchers* have more in common than those of *strategies*. The later are safer, and in a sense easier, to program, because they do not interact with FOL.

4.5.2.1. The master routines.

Calls to the executers of *tactics* and *matchers* are mediated by *TRYING*, and they *cannot* be called directly by any other routine. The MLISP code of *TRYING* follows. When the tactic is called from the *TRY* command, the whole expression returned by the *parser* will be passed as the argument *X* to *TRYING*. The conventions for this expression were outlined in the previous sections.

```
EXPR TRYING(X,PSWT,THREAD,PREP);
     BEGIN NEW OLDVL,REAS,G;
     G ← goal(THREAD);
     IF PREP THEN PREPARE(G,PSWT);
     IF REAS ← APPLY(GET(CAR X,'EXECUTER),THREAD CONS (PSWT CONS CDR X))
        THEN  IF REAS = T   THEN  MATCHWORK(THREAD,PSWT,CAR PROOF)
              ELSE  CURRENTGOALTHREAD ← THREAD
              ALSO udreason(G,REAS);
     RETURN REAS;
     END;
```

The executers of *strategies* can be called directly by another strategy, or recursively by itself. Thus the user does not need to call *TRYCMPL*. The *TRY*, command, however, uses *TRYCMPL* in order to mediate calls to the executers of strategies. The code of *TRYCMPL* follows.

```
EXPR TRYCMPL(X,PSWT,THREAD:G,REAS);
    IF REAS←APPLY(GET(CAR X,'EXECUTER),THREAD CONS (PSWT CONS CDR X))
      THEN IF PROVED(G←goal(THREAD))
                THEN RPLACD(CDR G,'PROVED? ? BY?  CONS REAS)
      ELSE REAS;
```

The code of these two routines was given here only for ease of reference. The user need not be concerned with this code, but looking at it may make it easier to understand the conventions outlined in this chapter.

4.6.2.2. The expression returned by executers.

The expression returned by executers of *tactics* and *matchers* is of paramount importance. It must obey the following rules. Failure to follow these rules will cause fatal errors.

1) NIL must be returned if the tactic could not be applied to the goal, or if the matcher failed. In this case, *nothing* happens to the goal structure. It is a "no-operation".

2) The LISP atom T must be returned if a match occurred; this condition applies to the matchers and to some tactics that sometimes match a goal (i.e., *REWRITE*, *SIMPLIFY*).

3) In the case of successful subgoal creation by a tactic, the expression returned must be the *REASON*: this expression is going to be stored as the *goal REASON*, by the master routine *TRYING*, and the *unwinder* of the tactic will use this information at a later time. (The user that programs a new tactic has complete freedom to choose this expression, as long as it is neither NIL nor T. The unwinder must be designed accordingly.)

In the case of *strategies*, the returned expression is not of the same importance. Only minor errors will result from returning a different expression. However, in order for error messages to work properly, it is convenient to return NIL if the strategy did not achieve *anything* at all (i.e., *no tactic or matcher could be successfully applied*), and otherwise a quoted expression like the name of the strategy. This quoted expression will be used as follows by TRYCMPL: if the goal was proved, it will append the information: "PROVED BY " followed by the quoted expression, and the only effect will be its appearance when the user displays the goal with the *SHOWGOAL* command.

4.6.2.3. Executers of tactics.

The executer of a tactic must update the goal structure by adding the newly created subgoals, as *sons* to the goal being tried.

The addition of subgoals is accomplished by invoking the routine *ADDSUBGOALS*; the first argument passed to this routine must be the thread of the goal being tried, and the second must be the number of sons to be created.

Most of the parts attached to goals are passed down, hereditarily, to their sons which are created by *ADDSUBGOALS*. But the goalwff must be updated, in every case, using the macro *udgoalwff*. Some tactics update other parts, for instance: the quantifier rules update the *quantelimlist*, and the tactic ⊃I updates the factlist.

We shall see here just two simple examples: the executers for ∧I and ∨I.

```
EXPR TRYAND(THREAD,PSWT:G,W);
 IF MAINSYM(W←goalwff(G←goal(THREAD)))='?∧
  THEN ADDSUBGOALS(THREAD,2)
  ALSO udgoalwff(son(1,G),LFAND(W))
  ALSO udgoalwff(son(2,G),RTAND(W))
  ALSO (IF PSWT THEN PRINTDESC(THREAD))
  ALSO RETURN(<'?∧I >);


EXPR TRYOR(THREAD,PSWT,SP:G,W);
 IF MAINSYM(W←goalwff(goal(THREAD)))='?∨
  THEN G←ADDSUBGOALS(THREAD,1)
  ALSO udgoalwff(G,IF SP=1 THEN LFOR(W) ELSE RTOR(W))
  ALSO (IF PSWT THEN PRINTDESC(THREAD))
  ALSO RETURN(<'?∨I ,SP>);
```

We shall see the *unwinders* of these two tactics in the section on *unwinders*. The *unwinder* UNWOR will use the information stored in the goal *REASON* by TRYOR.

4.6.2.4. Executers of matchers.

The executer of a matcher must call some FOL decision procedure; if the procedure decides that the *WFF* of the goal is *TRUE*, then we have a *match*. In this case the matcher must add the *WFF* as a new *VL* to the FOL *PROOF*, and then return the atom *T*. At this point, the last line of the *PROOF* must be the *VL* that matched the goal being tried.

These operations are done by calling the FOL routines that create *VLs*. This requires some understanding of the FOL code that goes beyond the scope of this chapter[4].

For the sake of completeness we show here the code of a matcher, that combines both *TAUT* and *TAUTEQ*. It attempts to match the goal *WFF* against the list of *facts* of the goal plus any list of *VLs* given by the user when calling this matcher using the *TRY* command. The user given *VLs* are passed in the parameter *VLLIST*. *TAUTMNG* is a FOL routine that decides tautologyhood, and *NEWSTEP* is the FOL routine that creates a new *VL*; care must be exercised when using *NEWSTEP*, because there are several ways of invoking it depending on the expressions returned by the different FOL decision procedures.

```
EXPR TRYTAUT(THREAD,PSWT,A,VLLIST,TEST); %TEST  to see if apply EQUTEST2 %
% A is 3 or 4 depending on whether TAUT or TAUTEQ is to be used. However,
            if A is 5 then both TAUT and TAUTEQ are tried. %
  BEGIN NEW W,X,G,AL;
   W←goalwff(G←goal(THREAD));
   IF AL←facts(G) THEN AL←CDR AL;
```

---

[4] Admittedly this is not an ideal situation. But we shall elaborate on the remedies in the final chapter of this thesis.

```
IF TEST ∧ (A=4) ∧ (¬EQUTEST2(W CONS AL)) THEN A←3;
IF X←TAUTMNG(A, W,?*APPEND(AL,VLLIST)) THEN NEWSTEP(X)
                                ALSO RETURN(T);
IF PSWT THEN TRYTAUTMSG(A) ALSO RETURN NIL;
END;
```

4.5.2.5. *Executers of strategies.*

The executer of a strategy sequences the calls to the executers of other operative elements.

It calls the executers of tactics and matchers indirectly, but those of strategies directly. In each case, the user must make sure that the appropriate arguments are being passed to each executer.

4.5.2.5.1. Example: elimination.

```
EXPR TRYELIM(THREAD,PRINTSWITCH,DEPTH);
IF ?*GREAT(DEPTH,0) THEN
  BEGIN NEW S,DESC,G;
  IF ((S←MAINSYM(goalwff(G←goal(THREAD)))) ∈ <'?∀ ,'?∃ ,'?∧ ,'?⊃ ,'?≡ >)
       THEN S←GET(S,'TACTICALL)
       ELSE IF S←EXISTORASSU(G,'?∨ ) THEN S←<'CASES ,S>
       ELSE RETURN NIL;
  IF TRYING(S,PRINTSWITCH,THREAD,T)
              THEN  DESC ← REVERSE(descendants(G))
              ALSO BEGIN
                L; TRYELIM(CAR(DESC) CONS THREAD,PRINTSWITCH,DEPTH-1);
                   IF DESC←CDR DESC THEN GO L;
                   END
              ALSO  RETURN('ELIM )
  ELSE RETURN NIL;
  END;
```

Notice that the strategy does not reset any of the defaults. For this particular strategy, the calls to the executers of the five tactics: ∀I, ∃I, ∧I, ⊃I, and ≡I, have been attached to LISP atoms in order to make the code compact. The GOAL routine *EXISTORASSU* determines whether a goal has a disjunction among its *facts*.

First it is determined whether elimination can be applied any further, and the appropriate calling expression is assigned to the variable S. The appropriate tactic is then called, and TRYELIM calls itself recursively on the sons, thus expanding the tree depth first.

4.5.2.5.2. Example: LOGIC.

We also show here the code of *LOGIC*, without comments. This strategy will be discussed extensively and a higher level description of its heuristics will be presented in the chapter on automatic theorem proving.

```
EXPR TRYLOGIC(THR,PSWT,FCL);
BEGIN NEW THRQUEUE,G,THREAD,FAILQ,PASS,PREP,S,D,TEMP,N,I,SUC,TH1,OLFQ,LFQ;
G←goal(THREAD←THR); PASS←1;OLFQ←0;
IF FCL THEN addonefact(G,'PROVED CONS FCL);
SRCH; IF MATCHSEARCH(THREAD,T) THEN GO MTCH;
IF T=(PREP←TRYING(<'SIMPLIFY,NIL>,PSWT,THREAD,NIL)) THEN GO MTCH
      ELSE IF PREP THEN PREP←NIL ALSO GO DS;
IF simpsetaddflag(G) THEN
    IF T=(PREP←TRYING(<'REWRITE,T,NIL,<<NIL,NIL>>>,PSWT,THREAD,NIL)) THEN GO MTCH
      ELSE IF PREP THEN PREP←NIL ALSO GO DS;
GRIND; PREP←NIL;
      IF ¬(X←TRYING(<'SIMPLIFY,NIL>,PSWT,THREAD,NIL)) THEN
      IF QUANT(S←MAINSYM(goalwff(G)))
            THEN TRYING(GET(S,'TACTICALL),PSWT,THREAD,NIL)
      ELSE IF simpsetaddflag(G)
                ∧ (X ← TRYING(<'REWRITE,T,NIL,<<NIL,NIL>>>,PSWT,THREAD,NIL))
                      THEN ( IF X=T THEN GO MTCH )
      ELSE
           ( IF S = '?⊃  THEN PREP←T  ALSO S←<'?⊃I>
             ELSE IF S ∈ <'?∧ ,'?≡ >        THEN S←GET(S,'TACTICALL)
             ELSE IF S←EXISTORASSU(G,'?∨ ) THEN S←<'CASES ,S>
             ELSE  FAILQ←THREAD CONS FAILQ ALSO GO L2 )
        ALSO TRYING(S,PSWT,THREAD,NIL)
     ELSE IF X=T THEN GO MTCH;
DS; N←LENGTH(D←descendants(G)); I←1 ; SUC←NIL; TEMP←NIL;
LUP; S←son(I,G);
   IF MATCHSEARCH(TH1←S CONS THREAD,PREP) THEN SUC←T
                              ELSE TEMP←TH1 CONS TEMP;
   IF ?*GREAT(N,I) THEN I←I+1 ALSO GO LUP;
   IF SUC THEN IF ?*GREAT(N,2)
            THEN THRQUEUE←?*APPEND(TEMP,THRQUEUE)
            ELSE GO MTCH
      ELSE THRQUEUE←?*APPEND(THRQUEUE,TEMP);
L2; IF THRQUEUE THEN THREAD←CAR THRQUEUE ALSO THRQUEUE←CDR THRQUEUE
   ALSO IF UNTRIED(G←goal(THREAD)) THEN IF ATOM(addedfacts(G)) THEN GO GRIND
                                          ELSE GO SRCH
            ELSE GO L2;
TERPRI();
IF NULL(FAILQ) THEN PRINC("Strange behavior of LOGIC: failqueue is empty!")
            ALSO RETURN NIL;
PRINC("We have a failqueue of length:  ");PRINC(LENGTH(FAILQ));
TEMP←NIL; PASS←PASS+1;
L3; IF FAILQ THEN THREAD←CAR FAILQ ALSO FAILQ←CDR FAILQ
      ALSO  ( IF UNTRIED(G←goal(THREAD))
                  THEN IF addedfacts(G) THEN THRQUEUE←THREAD CONS THRQUEUE
                        ELSE TEMP←THREAD CONS TEMP )
```

```
        ALSO GO L3;
TERPRI();
IF OLFQ=(LFQ←LENGTH(THRQUEUE))
        THEN PRINC("Failure: can't prove anything on failqueue.") ALSO RETURN NIL
        ELSE PRINC("Starting a new ")
        ALSO PRINC(PASS) ALSO PRINC("-th pass on new queue of length: ")
        ALSO PRINC(OLFQ←LFQ) ALSO FAILQ←TEMP ALSO GO L2;
MTCH; IF EMPTYTHREAD(NEXTGOALTHREAD) THEN TERPRI() ALSO PRINC("LOGIC SUCCEEDED!")
        ELSE G←goal(THREAD←NEXTGOALTHREAD)      ALSO GO SRCH;
RETURN 'LOGIC CONS FCL;
END;
```

The routine *MATCHSEARCH* is used by *LOGIC* to try out all match possibilities. However, the *MONADIC* matcher could be tried against some of the facts. This is not done because it often causes the system to run out of storage space.

```
EXPR MATCHSEARCH(THREAD,PREP:W);
  TRYING(<'UNIFY ,NIL>,NIL,THREAD,PREP)
      v TRYING(<'TAUT , 3, NIL,NIL>,NIL,THREAD,NIL)
      v  (IF MONASFLAG ∧ QUICKTEST(W←goalwff(goal(THREAD)),NIL)
            THEN TRYING(<'MONADIC,NIL,T,NIL>,NIL,THREAD,NIL) )
      v TRYING(<'EQUNIFY>,NIL,THREAD,NIL) ;
```

In order to make this example complete, we also show the code of the corresponding parser.

```
EXPR PARSELOGIC(G);
IF TK2@('LOGIC,'logic) THEN
     < 'LOGIC ,IF TK2@('PLUS,'plus) THEN VLLIST#()>;
```

4.5.3. Unwinders.

Only tactics have unwinders. The unwinder reads the unwinding information stored in the goal *REASON*, obtains the *VLs* of the proven *sons* of the goal, reconstructs the expression that must be passed to *NEWSTEP* in order for the new *VL* to be created, and returns this expression without calling *NEWSTEP*.

The master routine *UNWIND* controls unwinders, and this master routine is going to pass the returned expression to *NEWSTEP*. Thus, the convention to be followed is that the unwinder returns the expression that needs be passed to *NEWSTEP* in order for the new proof step to be created.

As examples, we show the code of the unwinders of the tactics: ∧I, and ∨I. In these examples, the macro *vlofpg* accesses the *VL* of a *proved goal*. The second example illustrates the use of the goal *REASON*, which is accessed with the macro *reason(G)*, in order for the unwinder to obtain the unwinding information.

The rather incomprehensible code of both unwinders is due to the FOL system. We shall elaborate more on this problem in the conclusion of this thesis.

```
EXPR UNWAND(G);
      BEGIN NEW X,Y;
      DEPLIST←DEPOF(X←vlofpg(son(1,G))) UNION2 DEPOF(Y←vlofpg(son(2,G)));
      RETURN (<goalwff(G),<THISLINE,'∧I? ,<'LIST&,CAR(X),CAR(Y)>>>);
      END;


EXPR UNWOR(G);
      BEGIN NEW X,W;
      DEPLIST←DEPOF(X←vlofpg(son(1,G))) ;
      W←goalwff(G);
      RETURN (<goalwff(G),<THISLINE,'∨I? ,
            IF CADR(reason(G))=1 THEN <'OI&, NUMOF(X),'WFF& CONS RTOR(W)>
            ELSE <'OI&,'WFF& CONS LFOR(W),NUMOF(X)>>>);
      END;
```

4.6. Introducing a new element to GOAL.

After programming a new operative element, the user must *introduce* it to the system and then load the new routines.

The *introduction* is accomplished by calling a GOAL routine that makes the components of the element known to GOAL.

The following three examples are self-explanatory.

```
NEWTACTIC ( '?∧I , 'PARSAND , 'TRYAND , 'UNWAND ) ;

NEWMATCHER( 'TAUT , 'PARSETAUT , 'TRYTAUT ) ;

NEWSTRATEGY( 'LOGIC , 'PARSELOGIC , 'TRYLOGIC ) ;
```

In each case the first argument is the *standard name*. The associated routines will be stored in the property list of that atom.

4.7. Conclusion.

In this chapter we have presented some documentation for the would-be hackers of GOAL. We may conclude that the programming of new *strategies* should be encouraged, while the programming of new *tactics* and *matchers* will remain, for the time being, a ground reserved for hardy souls. This situation may change when FOL has been redesigned, as we shall attempt to show in the final chapter of this thesis.

We conclude this chapter with a summary of the expressions needed in order to call the presently available tactics and matchers, for easy reference for programmers of strategies.

4.7.1. Summary of calls to tactics and matchers.

Calling the executer of a tactic or matcher, from the executer of a strategy, must be always done by calling *TRYING*, which takes four arguments. The first of these four is itself a list whose first element is the *standard name* of the callee. Any parameters that are specific to an operating element must also be passed as part of this list.

For easy reference we shall now list the ways to use the first argument, for the most common tactics and matchers. The second argument is a printswitch (normally T), the third one is the *thread* of the goal, and the fourth is a switch that should normally be T.

Thus the most usual, and simplest way to call them is:

TRYING (EXPRESSION , T , THREAD , T)

where EXPRESSION is as follows:

<'?∧| >

<'?⊃| >

<'?≡| >

<'?∀| ,NIL>

<'?∃| ,NIL>

<'SIMPLIFY,NIL>

```
<'REWRITE,T,NIL,<<NIL,NIL>>>

<'CASES, NIL>

<'UNIFY ,NIL>

<'MONADIC,NIL,T,NIL>

<'EQUNIFY>

<'TAUT , 3, NIL,NIL>
```

Some other calls are valid. In particular, in the call to *TAUT*, 3 can be replaced by 4 to invoke *TAUTEQ*, or by 5 to invoke both *TAUT* and *TAUTEQ*. In the call to *CASES*, NIL can be replaced by a pointer to a *VL*, and in the call to *REWRITE* the expression <<NIL,NIL>> can be replaced by a *simpset*.

Also, in the calls to *TAUT* and to *MONADIC* the first NIL can be replaced by a list of *VLs*, and in that to *UNIFY* it can be replaced by a pointer to a *VL*.

Though some other variations are also possible, the above list should take care of most needs.

5. AUTOMATIC THEOREM PROVING IN GOAL.

Our research fringes on the area of automatic theorem proving, but differs in its spirit from most of the current research in that discipline.

Whereas research in automatic theorem proving typically is machine oriented, and is concerned with obtaining proofs efficiently by careful management of the available resources, ours is strictly based on heuristic sequencing of natural deduction rules for the First Order Predicate Calculus.

One consequence of this approach is that, when a theorem is proved by a strategy, a complete FOL proof of that theorem is produced, which the user can inspect and understand. This differs from the situation, common to many theorem provers, in which it is often very difficult to understand how a particular theorem was concluded to be valid by the machine.

Although we are only secondarily concerned with theorem proving, some effort was invested in devising a heuristic that would be a powerful theorem prover of its own. This is the strategy LOGIC, presented in the next section.

The effort to augment the power of LOGIC has forced us to deal with some unsolved issues of current interest to researchers in automatic theorem proving. One of the purposes of this chapter is to contribute our experience to these discussions.

5.1. Automatic theorem proving by LOGIC.

The routine LOGIC combines all the simple (or atomic) tactics and matchers available to date in GOAL. This section comments LOGIC in plain English, and the next section gives a an algorithmic summary description of it. The reader may be well advised to read both descriptions in parallel.

LOGIC expands the tree of sub-goals in *breadth first* manner, using a *queue* of unproved sub-goals. The reason for the breadth first scheme is that in many cases the system is unable to match sub-goals that have been decomposed too far down. Since a proved sub-goal is frequently used in the proof of a descendant of one of its *brothers*, a *depth first* heuristic fails in those cases where the "wrong" branch of the tree was decomposed first. This happens in the pair example shown next; there, a depth first version of LOGIC (with which I experimented first) succeeded only when the two conjuncts were given in the "correct" order; whereas the presently implemented version succeeds either way.

At every node, LOGIC first attempts to match it using all the different matchers available: UNIFY, TAUT or TAUTEQ and MONADIC; unification is attempted against every fact in the attached *FACTLIST*; TAUT or TAUTEQ is called against the whole collection of facts; MONADIC is, at present, called only against the *GOALWFF* alone, because calling it against a whole set of *VL*s dramatically slows down the system and it often causes the available storage capacity to be exceeded.

If the *GOALWFF* is matched and the goal is not a top level one, the system looks up the other descendants of the parent of the matched goal, i.e. it attempts unwinding the proof as far up as possible, until it either proves a top level goal or it finds one or more unproved sons of a parent of a just proved goal. In this event, it adds the just proved goal to the *FACTLIST* of the unproved descendants of its parent, at all levels in these branches. It also places any unproved leaves of these branches in front of the queue (so they will be tried next) because they stand a better chance now that a new fact has been added to their *FACTLISTs*.

If no match is obtained, LOGIC checks whether anything has been added to the *SIMPSETLIST* of that node[1] since the last attempt at rewriting that goal or any one of its ancestors. If this is the case, it attempts to *rewrite* the goal. If the *GOALWFF* rewrites to *TRUE*, this is treated as a match, as described above. If the *WFF* rewrites to a different *WFF*, a son to that goal is created and is treated as described below. If the *WFF* does not rewrite, then other tactics will be tried in the following order.

Now LOGIC first looks up whether the main symbol of the goalwff is $\forall$, $\exists$, $\wedge$, $\equiv$ or $\supset$. In these cases it calls the corresponding tactic, thus generating one or more sons to that goal. If this is not possible, it looks up whether there is any *fact* in the *FACTLIST* that is a disjunction; if so, CASES is applied against that *fact*.

If none of the attempts to either match or decompose the goal succeeded, the goal is placed on a list of *failed* goals.

If a successful decomposition is obtained, LOGIC immediately tries to match each one of the just created sons. If a match (or perhaps more than one) is obtained, any unproved siblings of the matched goal will be placed in front of the queue for the same reason mentioned earlier. If none matches, they are all placed at the *end* of the queue.

After this, LOGIC picks the first goal in the queue and repeats the whole process just described, with one variation: since an attempt to match is made before placing a goal in the queue, no new attempt is now made unless some *fact* has been added to the goal (as a consequence of having proved a brother of some ancestor since it was placed in the queue). It may also be the case that the goal was in the meantime tried or perhaps even proved (and perhaps also "garbage collected" from the tree), because after a match unproved brothers are put in front of the queue. The system is able to recognize all these situations and treat them properly.

Now, what happens if the queue becomes empty? There must be some goals in the *fail list*, or otherwise LOGIC would have already proved a top level goal by now. All the goals in the *fail list* are examined; if any of them have experienced any change since they were placed there (i.e. additions of *facts* to them), these are placed in the queue of goals to be tried, and the whole process continues. This does not cause an infinite loop, because every time that the queue of goals to be tried becomes empty, LOGIC checks whether any changes to the list of failed goals have occurred. If there is no change, it exits, leaving the tree in the state it has gotten to, and announcing to the user the number of unproved leaves in the tree.

A successful exit occurs only when a top level goal is reached. If the original call to LOGIC

---

[1] *The SIMPSETADDFLAG is used for this purpose*

by the user was on a goal that is not top level, LOGIC will work below that node only as long as the node does not become proved. But, if it succeeds in proving it, instead of exiting it will continue working to its parent and down to its unproved brothers.

6.1.1. Summary of the LOGIC heuristics.

1.- Attempt MATCHING. If it succeeds then go MATCH.

TRYING: 2.- If SIMPSETADDFLAG, attempt REWRITING.
If it rewrites to TRUE, then go MATCH.
If it rewrites to a different WFF, then go SPLIT.

3.- Attempt one of the tactics: ∀I, ∃I, ∧I, ≡I, or ⊃I.
If one of these succeeds, then go SPLIT.

4.- Attempt CASES. If it succeeds then go SPLIT.

FAIL: 5.- Place goal in FAIL list. Go 7.

SPLIT: 6.- For each one of the sons, try MATCHING it.
If none matched, then place them at the
end of the QUEUE of goals to be tried.
If there is a match, then,
(If there are more than one still unmatched sons,
then place them in front of the QUEUE and
go NEXT, else go MATCH).

7.- If QUEUE is empty then go 9.

NEXT: 8.- Pick first element of QUEUE. Attempt MATCHING.
If match succeeds then go MATCH else go TRYING.

9.- Have facts been added to any goals in the FAIL list?
If yes, place them in the QUEUE and go NEXT.
If no, EXIT (failure).

10.- If NULL(NEXTGOALTHREAD) then EXIT (success),
else place NEXTGOALTHREAD in front of the QUEUE and go NEXT.

Notice that after any match in the goal structure the global variable NEXTGOALTHREAD will be pointing to some unproved descendant of the parent of some just proved goal, unless a top level goal was proved, in which case the variable will be NIL.

5.1.2. The PAIR example.

The following example is interesting on several accounts. It illustrates the following features of GOAL.

1. A proved subgoal is required in order to prove its *brother*; GOAL attaches it to this brother, and to any one of its descendants; thus one proved subgoal *fertilizes* another branch of the goal tree.

2. This proved subgoal is included in the simpset of the other branch, because it is a universally quantified equivalence.

3. *Conditional simplification.* LOGIC would not succeed without this feature, although a different heuristic would. However, that different heuristic would not succeed in the examples from [Kelley 1955], whereas LOGIC does.

4. The use of the *quantelimlist*. Its effect is similar to *Skolemization*.

5.1.2.1. Statement of the problem.

Given the axiom of EXTENSION, which states that two sets are equal if and only if they have the same elements, and the PAIR axiom, which states the existence of the *unordered pair of x and y* (i.e., a set whose only elements are x and y), the goal is to prove that the unordered pair is *unique*.

LOGIC generates an eight step proof in FOL automatically. This proof is more compact than what most sophisticated FOL users would normally achieve.

5.1.2.2. The GOAL generated proof.

The complete dialogue between the user and the system follows. Five asterisks is the FOL prompt. User given commands begin immediately after the prompt and end with the first semicolon or double semicolon. Anything else is typed by either FOL or GOAL. As an exception to the above rule, the FOL command *SHOW PROOF* generates a type out of the complete FOL proof, in which many lines beginning with five asterisks are typed by FOL (not by the user); these lines indicate the *reason* for the next line of the proof, i.e. how that line was obtained in FOL. Reasons generated by the GOAL unwinding mechanism are indistinguishable from those that would result from direct use of FOL for interactive construction of the same proof.

```
*****DECLARE INDVAR x y z z1 u v w;

*****DECLARE PREDCONST ∈ 2 [INF];

*****AXIOM EXTENT: ∀x y.(x=y≡∀u.(u∈x≡u∈y));;

EXTENT: ∀x y.(x=y≡∀u.(u∈x≡u∈y))

*****AXIOM PAIR: ∀x y.∃w.∀u.(u∈w≡(u=x∨u=y));;

PAIR: ∀x y.∃w.∀u.(u∈w≡(u=x∨u=y))

*****GOAL ∀x y.∃z.(∀w.(w∈z≡(w=x∨w=y))∧∀z1.(∀w.(w∈z1≡(w=x∨w=y))⊃z1=z))
                                                    ASSUME PAIR SASSUME EXTENT;

Goal #1: ∀x y.∃z.(∀w.(w∈z≡(w=x∨w=y))∧∀z1.(∀w.(w∈z1≡(w=x∨w=y))⊃z1=z))

*****SHOWGOAL;

Goal #1: ∀x y.∃z.(∀w.(w∈z≡(w=x∨w=y))∧∀z1.(∀w.(w∈z1≡(w=x∨w=y))⊃z1=z))
    VLSASSU:  EXTENT ∀x y.(x=y≡∀u.(u∈x≡u∈y))
    VLASSU:   PAIR ∀x y.∃w.∀u.(u∈w≡(u=x∨u=y))
      Simpsets:  ( BY  LOGICTREE COMPTREE)
```

COMMENT: *the showgoal command shows that the axiom of EXTENT has been added as an assumption and the axiom of PAIR as a sassumption. It also shows that (by default) the simpsets LOGICTREE and COMPTREE have been attached. Next we show the result of invoking the LOGIC tactic, and the proof it generates. A commentary follows.*

```
*****TRY USING LOGIC;

Goal #1#1: ∀x y.∃z.(∀w.(w∈z≡(w=x∨w=y))∧∀z1.(∀w.(w∈z1≡(w=x∨w=y))⊃∀u.(u∈z1≡u∈z)))
Goal #1#1#1: ∃z.(∀w.(w∈z≡(w=x∨w=y))∧∀z1.(∀w.(w∈z1≡(w=x∨w=y))⊃∀u.(u∈z1≡u∈z)))
Goal #1#1#1#1: ∀w.(w∈z≡(w=x∨w=y))∧∀z1.(∀w.(w∈z1≡(w=x∨w=y))⊃∀u.(u∈z1≡u∈z))
Goal #1#1#1#1#1: ∀w.(w∈z≡(w=x∨w=y))
Goal #1#1#1#1#2: ∀z1.(∀w.(w∈z1≡(w=x∨w=y))⊃∀u.(u∈z1≡u∈z))

1 ∃z.∀w.(w∈z≡(w=x∨w=y))

2 ∀w.(w∈z≡(w=x∨w=y))  (2)

3 ∀z1.(∀w.(w∈z1≡(w=x∨w=y))⊃∀u.(u∈z1≡u∈z))  (2)

4 ∀w.(w∈z≡(w=x∨w=y))∧∀z1.(∀w.(w∈z1≡(w=x∨w=y))⊃∀u.(u∈z1≡u∈z))  (2)

5 ∃z.(∀w.(w∈z≡(w=x∨w=y))∧∀z1.(∀w.(w∈z1≡(w=x∨w=y))⊃∀u.(u∈z1≡u∈z)))

6 ∀x y.∃z.(∀w.(w∈z≡(w=x∨w=y))∧∀z1.(∀w.(w∈z1≡(w=x∨w=y))⊃∀u.(u∈z1≡u∈z)))

7 ∀x y.∃z.(∀w.(w∈z≡(w=x∨w=y))∧∀z1.(∀w.(w∈z1≡(w=x∨w=y))⊃z1=z)) ≡
                    ∀x y.∃z.(∀w.(w∈z≡(w=x∨w=y))∧∀z1.(∀w.(w∈z1≡(w=x∨w=y))⊃∀u.(u∈z1≡u∈z)))
```

8 ∀x y.∃z.(∀w.(w∈z≡(w=x∨w=y))∧∀z1.(∀w.(w∈z1≡(w=x∨w=y))⊃z1=z))

LOGIC SUCCEEDED!

COMMENT: *everything following the command "TRY USING LOGIC" has been typed by the system. The first subgoal that matches is #1#1#1#1#1 (line 2), and #1#1#1#1#2 matches immediately thereafter (line 3). Line 3 depends on (2) because the system added (2) to the simpset attached to goal #1#1#1#1#2 and it actually used line (2) to prove this goal. Finally we use the FOL "SHOW PROOF" command to display the proof produced by the logic tactic.*

*****SHOW PROOF;

*****UNIFY PAIR;

1 ∃z.∀w.(w∈z≡(w=x∨w=y))

*****∃E ↑ z;

2 ∀w.(w∈z≡(w=x∨w=y)) (2)

*****REWRITE ∀z1.(∀w.(w∈z1≡(w=x∨w=y))⊃∀u.(u∈z1≡u∈z))
BY ↑ EXTENT LOGICTREE COMPTREE;

3 ∀z1.(∀w.(w∈z1≡(w=x∨w=y))⊃∀u.(u∈z1≡u∈z)) (2)

*****∧I (2 3);

4 ∀w.(w∈z≡(w=x∨w=y))∧∀z1.(∀w.(w∈z1≡(w=x∨w=y))⊃∀u.(u∈z1≡u∈z)) (2)

*****∃I ↑ z ;

5 ∃z.(∀w.(w∈z≡(w=x∨w=y))∧∀z1.(∀w.(w∈z1≡(w=x∨w=y))⊃∀u.(u∈z1≡u∈z)))

*****∀I ↑ x y;

6 ∀x y.∃z.(∀w.(w∈z≡(w=x∨w=y))∧∀z1.(∀w.(w∈z1≡(w=x∨w=y))⊃∀u.(u∈z1≡u∈z)))

*****REWRITE ∀x y.∃z.(∀w.(w∈z≡(w=x∨w=y))∧∀z1.(∀w.(w∈z1≡(w=x∨w=y))⊃z1=z))
BY EXTENT LOGICTREE;

7 ∀x y.∃z.(∀w.(w∈z≡(w=x∨w=y))∧∀z1.(∀w.(w∈z1≡(w=x∨w=y))⊃z1=z)) ≡
∀x y.∃z.(∀w.(w∈z≡(w=x∨w=y))∧∀z1.(∀w.(w∈z1≡(w=x∨w=y))⊃∀u.(u∈z1≡u∈z)))

*****TAUT ∀x y.∃z.(∀w.(w∈z≡(w=x∨w=y))∧∀z1.(∀w.(w∈z1≡(w=x∨w=y))⊃z1=z)) 6,7;

8 ∀x y.∃z.(∀w.(w∈z≡(w=x∨w=y))∧∀z1.(∀w.(w∈z1≡(w=x∨w=y))⊃z1=z))

*****

6.1.3. Commentary to the PAIR example.

The steps followed by the LOGIC tactic are explained in detail in this section.

When LOGIC is invoked on the goal #1, it first attempts to match the goal; it fails. Then it attempts to rewrite the goal by syntactic simplification using the attached simpsets plus the *sassumed* axiom of extent. This produces a different *WFF*, which becomes goal #1#1. This *WFF* was obtained by rewriting "z1=z" by the axiom of extent. Why was "w=x∨w=y" not rewritten? The reason is that the rewrite tactic has noticed that this is part of the wff "w∈z≡w=x∨w=y" which has the same structure as a wff in the *assume list*: namely, it has the same structure, except for the leading quantifiers, as the *assumed* axiom PAIR. Recognizing that that part of the goal is potentially matchable against that fact, it does not rewrite it. This shows *conditional simplification*.

#1#1#1 is obtained from #1#1 by elimination of the leading universals.

The #1#1#1#1 is obtained by elimination of the leading existential.

This goal is then decomposed into two sub-goals because its main logical connective is "∧".

Next, #1#1#1#1#1 is unified by the UNIFY tactic against the axiom PAIR. This tactic recognizes that the wff "∀w.(w∈z≡(w=x∨w=y))" cannot be directly unified against PAIR, but that, by reintroducing the existential on z, which -as it *remembers*- was eliminated further up in the tree, the *WFF* "∃z.∀w.(w∈z≡(w=x∨w=y))" can be unified against PAIR. Thus it produces this *WFF* as a first line of the proof, and then it eliminates the existential, producing line 2 of the proof, which matches the subgoal. This matched subgoal is added as a *fact* to its brother #1#1#1#1#2[2].

When trying #1#1#1#1#2, LOGIC recognizes that a fact has been added to this sub-goal, namely line 2 which proves its brother. It first tries the matchers, which fail. When the goal is *prepared* by the first *TRY*, the system recognizes that the wff of the added fact, "∀w.(w∈z≡w=x∨w=y)" should be added to the simpset since it is a universally quantified equivalence. After the matchers fail, LOGIC recognizes that a new element has been added to the simpset. Therefore it attempts a new rewrite on this sub-goal. In this event, the wff rewrites to TRUE[3], thus this subgoal has been proved. Since it was the last unproved leaf of the tree, the proof now unwinds automatically.

If the two conjuncts of #1#1#1#1 had been switched as the goal was created (i.e. B∧A instead of A∧B), LOGIC would have produced exactly the same proof. This is noteworthy because proof of one of the two conjuncts is required in order to be able to prove the other. Thus a strictly top-down scheme would succeed only if the conjuncts were given in the "right" order. But LOGIC carries the search in a breadth first fashion.

---

[2] As the variable z is matched against the existentially eliminated variable in line 2, GOAL records this as a *binding*, meaning that now z, in the other branch of the goal tree, is not free any more for matching against arbitrary terms, as it was before. Also it records where in the goal tree that binding took place, and in case of an *abandoning* of an ancestor of that goal, z would be made free again.

[3] It does so because the simpset LOGICTREE is attached to the goal by default.

5.1.4. The initial theorems from Kelly.

LOGIC has generated automatic proofs of the first 32 theorems in the Appendix on Set Theory in [Kelley 1955]. It has also generated automatic proofs of several further theorems. In addition, for some theorems it has proved all but one of the subgoals it generated; in some of these cases the unproved subgoal could be proved by one additional FOL command, and then GOAL would unwind the proof.

[Kelley 1955] uses the following form of the comprehension axiom scheme:

COMPREHENSION: ∀x.(x∈{y|P(y)} ≡ SET(x) ∧ P(x)) .

Comprehension terms are automatically rewritten by LOGIC according to that axiom scheme. This is accomplished by the *simpset* COMPTREE, which is attached to goals by default.

The pattern of the proofs of those first 32 theorems is the same as in the following example (Theorem 4, part 2). In each case, the user has to attach the appropriate set of facts, using *SASSUME*, in order for LOGIC to succeed.

That pattern consists of a subgoaling by *REWRITE*, followed by a match by *MONADIC*.

5.1.4.1. An example from Kelly.

```
*****DECLARE INDVAR x y z;
*****DECLARE PREDCONST ∈ 2 [INF];
*****DECLARE OPCONST ∪ 2 [INF];
*****AXIOM SET: ∀x.(SET(x)≡∃y.x∈y);;
SET: ∀x.(SET(x)≡∃y.x∈y)
*****AXIOM UNION: ∀x y.x∪y={z|z∈x∨z∈y};;
UNION: ∀x y.(x∪y)={z|z∈x∨z∈y}
*****GOAL ∀x y z.(z∈x∪y≡z∈x∨z∈y) SASSUME SET UNION;
Goal #1: ∀x y z.(z∈(x∪y)≡(z∈x∨z∈y))
```

```
*****TRY USING LOGIC;

Goal *1*1:  ∀x y1 z.((∃y.z∈y∧(z∈xvz∈y1))≡(z∈xvz∈y1))

1 ∀x y1 z.((∃y.z∈y∧(z∈xvz∈y1))≡(z∈xvz∈y1))

2 ∀x y z.(z∈(xuy)≡(z∈xvz∈y))≡∀x y1 z.((∃y.z∈y∧(z∈xvz∈y1))≡(z∈xvz∈y1))

3 ∀x y z.(z∈(xuy)≡(z∈xvz∈y))

LOGIC SUCCEEDED!

*****SHOW PROOF;

*****MONADIC ;

1 ∀x y1 z.((∃y.z∈y∧(z∈xvz∈y1))≡(z∈xvz∈y1))

*****REWRITE ∀x y z.(z∈(xuy)≡(z∈xvz∈y))  BY  UNION SET LOGICTREE COMPTREE;

2 ∀x y z.(z∈(xuy)≡(z∈xvz∈y))≡∀x y1 z.((∃y.z∈y∧(z∈xvz∈y1))≡(z∈xvz∈y1))
*****TAUT ∀x y z.(z∈(xuy)≡(z∈xvz∈y)) 1,2;

3 ∀x y z.(z∈(xuy)≡(z∈xvz∈y))

*****
```

5.2. Issues in goal oriented theorem proving.

This chapter ends with a discussion of some problems for which we have not found any satisfactory solution. These have to do with some trade offs between the amount of manipulation of the assertions by theorem proving strategies and the complexity of these strategies.

Not having found one generally good way of dealing with these trade offs, perhaps the best approach would be to maximize the degree of user's control over the manipulations of the assertions in such a way that the strategies can control these manipulations with the same flexibility as they control the decomposition of goals.

This approach would be in keeping with the general conclusions suggested in this thesis. It is better to strive for a flexible environment in which strategies can be programmed, and to live with specialized ones, rather than with maximally powerful, heavy theorem provers. But a good deal of thought is still needed before the flexibility of GOAL can be extended to the manipulation of the assertions.

5.2.1. Subgoaling and assertions.

From our point of view as a user of GOAL, a goal is a *WFF* to be proved; attached to this *WFF*, there are *facts* or *assertions*, *simpsets*, and some other information. This approach is sensible because the reduction rules incorporated in the tactics are *natural*, in the sense that they correspond to the *natural deduction* [Prawitz 1965] rules of FOL. This makes it easy for the user to understand the description and to conduct interactive proof construction in GOAL.

From the point of view of the design of automatic theorem proving heuristics, the more elegant approach taken by [Brown 1977a, 1977b, 1978] is better. This researcher defines transformations between *sequents*, a *sequent* being a collection of *assertions* and *goals*. The meaning of a sequent is that the *disjunction* of the *goals* (i.e., at least one of them) follows from the *conjunction* of the *assertions*. That approach establishes a duality between *goals* and *assertions*, so that the rules that manipulate the latter do not have a different status from those that manipulate goals.

Our tactics can be described in that way, and most of the rules in Brown's papers are indeed in GOAL. However, in our system *goals* and *assertions* (or *facts*) have a quite different status because the latter are *VLs* of the FOL proof, while the *goals* are *WFFs* without any FOL status. In line with our efforts to keep GOAL consistent with FOL, any *WFFs* in the *FACTLIST*[4] of a goal are written as assumptions onto the FOL proof[5] before the goal is *tried*.

This stringent requirement that the *facts* must always be *VLs* makes *unwinding* simple.

5.2.2. Working on the assertions.

Some theorems can be proved from the axioms mainly by manipulation of the *goals*. In those cases, GOAL is generally successful. But some other theorems require many manipulations of the *axioms*, before these can be used to prove the goals. For instance: the axioms may be rewritten; conjunctive axioms may be decomposed into *VLs* that assert the disjuncts separately; or several axioms may be combined in order to obtain a different assertion.

5.2.2.1. RESOLVE.

In GOAL, some transformations of the *facts*, or *assertions*, have been built into the *prepare* mechanism. In particular, *PREPARE* attempts to *RESOLVE* an assumption generated by the *implication rule*, ⊃I, against the other *VLs* in the *FACTLIST*. *RESOLVE* is a FOL inference rule based on a variation of *UNIFY* that will perform some inferences of the following type: from

---

[4] For instance, the antecedent of an implication after subgoaling by the implication rule.

[5] Using the FOL *ASSUME* command.

two assertions "$\forall x.(A(x) \supset B(x)$" and "A(t)", an assertion "B(t)" may be inferred. *RESOLVE* is as yet undocumented, like *UNIFY*, and it is still in a developmental stage; thus we will not describe it any further.

For automatic theorem proving, it is often important that such inferences be drawn automatically. But the generation of many possible inferences from a set of facts tends to increase the complexity of the heuristics; it causes many new *VL*s to be generated, and in many cases it causes the theorem prover to fail because it takes a wrong path.

5.2.2.2. Rewriting assertions vs. conditional simplification.

In the PAIR example discussed earlier, we saw that *conditional simplification* prevented parts of the goal from being rewritten. If that part of the goal that claims the *existence* of the unordered pair had been rewritten, it would not have matched against the PAIR axiom that asserts its existence, unless that axiom had also been rewritten by the axiom of EXTENT.

This situation occurs quite often. One would be tempted to rewrite *every* assertion in the *FACTLIST* using the *simpset* attached to the goal, and to add the rewritten *VL*s to that *FACTLIST*, in order to increase the power of the theorem prover. But doing this would cause a large, probably exponential, increase in the running time, and it would heavily tax the storage requirements. Also it would cause many useless *VL*s to be added to the proof, and this would make the proofs generated by GOAL much more unreadable and difficult to understand.

Thus we chose not to rewrite assertions. Instead conditional simplification is used in order to prevent rewriting of those parts of goals that are potentially matchable against some *fact*.

Conditional simplification was implemented by testing, at each step in the (recursive) rewrite loop, whether the *subwff* would pass the *isomorphy* test that *UNIFY* uses. If it does, rewritting of that *subwff*, and of any one of its parts, is blocked.

This approach presents problems of its own, however. For instance, when there are some *facts* whose main connective is the equality symbol "="[6], the sides of any equality in the *GOALWFF* would not be rewritten. In many cases this restriction is excessive and it prevents effective theorem proving.

We have not found any near optimal solution to these trade offs. Instead, we have provided the *executer* of the *REWRITE* tactic with a flag to activate conditional simplification. When the tactic is called directly by the *TRY* command, the flag is off. In a user programmed strategy, the flag can be controlled by the strategy.

The interested reader may refer back to the discussion of conditional simplification in the introduction, where the difference between ours and the Edinburgh **[Gordon, Milner and Wadsworth 1977]** version was pointed out.

---

6 The equality symbol is a *predicate constant* in FOL.

6. SOME FUTURE ORIENTED CONCLUSIONS.

The three main accomplishments of our research are: the creation of a command language for *top down* construction of proofs in FOL and the demonstration of its usefulness; that this language is extensible; and the demonstration of the practibility of our approach to automatic theorem proving.

Enough has been said about these three aspects in this thesis. But not much has been said about what we have learned of how a first order logic proof checker *could* fit with a goal command language. Therefore we want to conclude with some remarks about this.

6.1. Ideal FOL and GOAL.

In an ideal FOL proof checker, the parsing of user's commands is completely separated from the "semantic" routines that effect the actions of these commands. The *parsing* routines and the *semantic* (or *action*) routines communicate through a carefully designed system of interfaces. Furthermore, the system of *reasons*[1] maps this system of interfaces so well that they could themselves be passed as input to FOL.

The first consequence of this is that the programming of new tactics and of new matchers in GOAL becomes as easy and reliable as the programming of strategies. At present, no faulty deduction by strategies is possible if the tactics and matchers are sound. Thus we can guarantee the user that extensions to GOAL will be *foolproof* if they are limited to the addition of strategies. With ideal FOL we can make the programming of new *tactics* and *matchers* foolproof as well.

For tactics, the programming of *unwinders* becomes unnecessary. They can be automatically generated. The user has to specify what FOL rule the tactic is inverse to, and to make sure that the *executer* returns an item that conforms to the rules for the FOL *reasons* for the proof steps. At unwinding time, FOL will then know how to take the appropriate action so as to generate the new step of the proof.

For matchers, the executer looks quite simple. It simply calls the appropriate FOL decision *procedure*, through the corresponding interface.

All of this is fairly obvious. FOL is not far from having this structure, but its actual code is not quite there yet.

---

[1] That is, the FOL reasons for the proof steps.

6.2. Extensibility and METAFOL.

Research on the formalization of FOL in FOL has progressed in parallel to our research **[Weyhrauch 1978a]**. That research aims at the mechanization of this formalization, so that properties of FOL can be both formalized *and* easily proved in FOL.

It has been already pointed out that a high level language for the programming of extensions by the user is both desirable and feasible. It appears that METAFOL offers both a language for describing new modes of inference and the possibility of proving their correctness in FOL. This seems a fruitful direction for further research. I think that it addresses the basic problems of describing new tactics, matchers, and strategies, adequately. Appropriate attention should also be given to the full range of questions regarding the type of facilities that ought to be given the user for manipulating *facts* and *simpsets* in the context of theorem proving strategies, in order to make a powerful and high level programming language for theorem proving applications.

This suggested research may bring about the exciting possibility that extensions to GOAL can be described *and* proved correct in FOL using METAFOL; when the user *convinces* FOL of the correctness of a proposed extension, it gets *accepted* and automatically converted into a working extension to GOAL.

7. APPENDIX 1: THE TAKEUCHI FUNCTION.

The following proof illustrates two aspects of GOAL: *top down* proof construction, and the use of extensibility. It is also interesting in that it shows the potential of First Order Logics for program verification.

7.1. Introduction.

The Takeuchi function was devised by Ikuo Takeuchi of the Electrical Communication Laboratory of Nippon Telephone and Telegraph Co. for the purpose of comparing the speeds of LISP systems. It can be made to run a long time without generating large numbers or using much stack. It is defined as follows.

$$tak(x,y,z) \leftarrow \text{if } x \leq y \text{ then } y \text{ else } tak(tak(x-1,y,z),tak(y-1,z,x),tak(z-1,x,y))$$

[McCarthy 1978a] showed that this function is equal to the following simpler expression.

$$tak0(x,y,z) = \text{if } x \leq y \text{ then } y \text{ else if } y \leq z \text{ then } z \text{ else } x$$

The same author [McCarthy 1978b] constructed a 50 step FOL proof of this fact, without using GOAL. We shall compare the proof using GOAL with McCarthy's proof.

7.2. A strategy for case analysis.

A strategy was added to GOAL for this proof. We shall use the example in order to illustrate the process of extending GOAL in detail. This strategy is very similar to IFCASES. It differs from it in that it does not expand the *conditional wffs* and the *conditional terms* into formulae without conditionals. Doing that expansion did not yield good results in this example. We chose the name *IFCASESHORT*, in order to distinguish it from *IFCASES*.

Let us first look at the parser. It was decided that the user had to explicitly tell to the parser the *WFF* on which the case analysis was to be carried. The following two routines implement the parser.

```
EXPR PARSEIFCASESHORT(G:X);
 IF TK2@('IFCASESHORT,'ifcaseshort)
      THEN IF X←WFF#(NIL) THEN RETURN(<'IFCASESHORT ,X>)
      ELSE IFCASEPARSEMSGH1() ALSO ENDL();

EXPR IFCASEPARSEMSGH1();
      BEGIN TERPRI();
      PRINC("IF-CASES-SHORT requires that you specify a WFF.");
      END;
```

Now let us look at the executer and comment the code. The parameter WF is the *WFF* specified by the user. The executer first calls the *CASES* tactic on WF and ¬WF. Thus, if the original *WFF* of the goals is GWF, the two subgoals generated by this tactic are: WF⊃GWF and ¬WF⊃GWF. For each of these two subgoals, our strategy will call the tactic ⊃I and then the *REWRITE* tactic. The calls to ⊃I occur at the label REP in the code, which is executed twice. A call to ⊃I causes the antecedent to be attached to the subgoal as an assumption. Since it will not necessarily be placed into the SIMPSET, the next three lines of code force this assumption to be written onto the FOL proof and to be put into the SIMPSET before calling REWRITE. The *prepare* mechanism causes a negation ¬WF to be also written as WF=FALSE because this form happens to work better with the *rewrite* code.

```
EXPR TRYIFCASESHORT(THREAD,PSWT,WF);
BEGIN NEW S, S1, MT, G, THR;
TRYING(<'CASES,WF>,PSWT,THREAD,T);
S←son(1,goal(THREAD));
S1←son(2,goal(THREAD));
REP; TRYING(<'?⊃I>,PSWT,S CONS THREAD,NIL);
PREPARE(G←goal(THR←NEXTGOALTHREAD),NIL);
MT←<<NIL,NIL>>;
VLADD(CAR PROOF,MT,'SUBSTLEAF&);
TRYING(<'REWRITE,NIL,<CAAR PROOF>,MT>,PSWT,THR,NIL);
IF S1 THEN S←S1 ALSO S1←NIL ALSO GO REP;
RETURN('IFCASESHORT );
END;
```

These routines must now be added to the system, together with the following statement.

```
NEWSTRATEGY('IFCASESHORT,'PARSEIFCASESHORT,'TRYIFCASESHORT);
```

And now the extension is complete. The most difficult part of this code is that which has to do with forcing the assumption into the SIMPSET. That part requires an understanding of the REWRITE code, which users of FOL cannot be required to possess. The example thus illustrates the importance of devising a high level language for programming strategies. That high level language should not be too restrictive in the amount of control allowed over assumptions, simpsets, *VL*s, and other items.

7.3. McCarthy's FOL proof.

The declarations, the axioms, and the whole proof devised by **[McCarthy 1978b]** follows. Axiom LESS comprises nine lemmas, not all of which are actually used in the proof. I found that LESS1, LESS3, and LESS6, are unnecessary. These lemmas are similar to the *verification conditions* used by program verification systems.

7.3.1. Declarations.

```
declare INDVAR x y z ∈ REAL;
declare OPCONST pred(REAL) = REAL[PRE];
declare OPCONST tak0(REAL,REAL,REAL) = REAL;
declare OPCONST tak1(REAL,REAL,REAL) = REAL;
declare PREDCONST <(REAL,REAL)[L←455,R←455];
declare PREDCONST ≤(REAL,REAL)[L←455,R←455];
```

7.3.2. Axioms.

```
LESS: LESS1: ∀x.pred x<x
      LESS2: ∀x.pred x≤x
      LESS3: ∀x y.((x<y∧(¬(x=y)∧¬(y<x)))∨((¬(x<y)∧(x=y∧¬(y<x)))∨(¬(x<y)∧(¬(x=y)∧y<x))))
      LESS4: ∀x y z.(((x<y∧y<z)⊃x<z)∧(((x≤y∧y<z)⊃x<z)∧(((x<y∧y≤z)⊃x<z)∧((x≤y∧y≤z)⊃x≤z))))
      LESS5: ∀x y.(x≤y≡(x<y∨x=y))
      LESS6: ∀x.¬(x<x)
      LESS7: ∀x.x≤x
      LESS8: ∀x y.(¬(x≤y)≡y<x)
      LESS9: ∀x y.(y<x≡¬(x≤y))

TAK0: ∀x y z.tak0(x,y,z)=IF x≤y THEN y ELSE IF y≤z THEN z ELSE x

TAK1: ∀x y z.tak1(x,y,z)=IF x≤y THEN y ELSE tak0(tak0(pred x,y,z),tak%
0(pred y,z,x),tak0(pred z,x,y))
```

7.3.3. The proof.

> "The proof is actually a cleaned up version of a 68 step proof that was in some ways more informative. Namely, I used the REWRITE rule with what inequalities I had and looked at the right hand side to see which ones I still should look for. In particular, the splitting of the main case into subcases was determined empirically by seeing what propositional terms appeared in the conditional expressions. From this point of view, FOL helped in generating the proof and didn't merely check a pre-existing proof."
>
> **[McCarthy 1978b]**

```
*****ASSUME x≤y;

1 x≤y  (1)

*****REWRITE tak1(x,y,z)=tak0(x,y,z) BY LOGICTREEU{ TAK0,TAK1,1};

2 tak1(x,y,z)=tak0(x,y,z)  (1)

*****⊃I ↑↑⊃↑;

3 x≤y⊃tak1(x,y,z)=tak0(x,y,z)

*****ASSUME ¬(x≤y);

4 ¬(x≤y)  (4)

*****REWRITE y<x BY LOGICTREEU{ LESS9,4};

5 y<x  (4)

*****ASSUME y≤z;

6 y≤z  (6)

*****ASSUME pred x≤y;

7 pred x≤y  (7)

*****∀E LESS2 y;
```

```
8 pred y≤y

*****∀E LESS4 pred y,y,z;

9 ((pred y<y∧y<z)⊃pred y<z)∧(((pred y≤y∧y<z)⊃pred y<z)
∧(((pred y<y∧y≤z)⊃pred y<z)∧((pred y≤y∧y≤z)⊃pred y≤z)))

*****TAUT pred y≤z 6,8:9;

10 pred y≤z  (6)

*****REWRITE tak1(x,y,z)=tak0(x,y,z) BY LOGICTREEu{ TAK0,TAK1,4:7,10};

11 tak1(x,y,z)=tak0(x,y,z)  (4 6 7)

*****⊃I 7⊃↑;

12 pred x≤y⊃tak1(x,y,z)=tak0(x,y,z)  (4 6)

*****ASSUME ¬(pred x≤y);

13 ¬(pred x≤y)  (13)

*****REWRITE tak1(x,y,z)=tak0(x,y,z) BY LOGICTREEu{ TAK0,TAK1,LESS7,4:6,10,13};

14 tak1(x,y,z)=tak0(x,y,z)  (4 6 13)

*****⊃I ↑↑⊃↑;

15 ¬(pred x≤y)⊃tak1(x,y,z)=tak0(x,y,z)  (4 6)

*****TAUT tak1(x,y,z)=tak0(x,y,z) 12,15;

16 tak1(x,y,z)=tak0(x,y,z)  (4 6)

*****⊃I 6⊃↑;

17 y≤z⊃tak1(x,y,z)=tak0(x,y,z)  (4)

*****ASSUME ¬(y≤z);

18 ¬(y≤z)  (18)

*****REWRITE z<y BY LOGICTREEu{ LESS9,18};

19 z<y  (18)

*****∀E LESS4 z,y,x;

20 ((z<y∧y<x)⊃z<x)∧(((z≤y∧y<x)⊃z<x)∧(((z<y∧y≤x)⊃z<x)∧((z≤y∧y≤x)⊃z≤x)))

*****∀E LESS5 z,x;
```

21 z≤x≡(z<x∨z=x)

*****TAUT z≤x 5,19:21;

22 z≤x (4 18)

*****VE LESS4 pred z,z,x;

23 ((pred z<z∧z<x)⊃pred z<x)∧(((pred z≤z∧z<x)⊃pred z<x)
∧(((pred z<z∧z≤x)⊃pred z<x)∧((pred z≤z∧z≤x)⊃pred z≤x)))

*****VE LESS2 z;

24 pred z≤z

*****VE LESS4 pred z,z,x;

25 ((pred z<z∧z<x)⊃pred z<x)∧(((pred z≤z∧z<x)⊃pred z<x)
∧(((pred z<z∧z≤x)⊃pred z<x)∧((pred z≤z∧z≤x)⊃pred z≤x)))

*****TAUT pred z≤x 22,24:25;

26 pred z≤x (4 18)

*****ASSUME pred x≤y;

27 pred x≤y (27)

*****ASSUME pred y≤z;

28 pred y≤z (28)

*****ASSUME ¬(pred x≤y);

29 ¬(pred x≤y) (29)

*****ASSUME ¬(pred y≤z);

30 ¬(pred y≤z) (30)

*****REWRITE tak1(x,y,z)=tak0(x,y,z) BY FOOu{ 27:28};

31 tak1(x,y,z)=tak0(x,y,z) (4 18 27 28)

*****REWRITE tak1(x,y,z)=tak0(x,y,z) BY FOOu{ 27,30};

32 tak1(x,y,z)=tak0(x,y,z) (4 18 27 30)

*****VE LESS4 pred x,z,y;

33 ((pred x<z∧z<y)⊃pred x<y)∧(((pred x≤z∧z<y)⊃pred x<y)
∧(((pred x<z∧z≤y)⊃pred x<y)∧((pred x≤z∧z≤y)⊃pred x≤y)))

```
*****∀E LESS5 z,y;
34 z≤y≡(z<y∨z=y)
*****TAUT ¬(pred x≤z) 19,29,33:34;
35 ¬(pred x≤z)  (18 29)
*****REWRITE tak1(x,y,z)=tak0(x,y,z) BY FOOU{ 28:29,35};
36 tak1(x,y,z)=tak0(x,y,z)  (4 18 28 29)
*****REWRITE tak1(x,y,z)=tak0(x,y,z) BY FOOU{ 29:30};
37 tak1(x,y,z)=tak0(x,y,z)  (4 18 29 30)
*****⊃I 28⊃31;
38 pred y≤z⊃tak1(x,y,z)=tak0(x,y,z)  (4 18 27)
*****⊃I 30⊃32;
39 ¬(pred y≤z)⊃tak1(x,y,z)=tak0(x,y,z)  (4 18 27)
*****TAUT tak1(x,y,z)=tak0(x,y,z) 38:39;
40 tak1(x,y,z)=tak0(x,y,z)  (4 18 27)
*****⊃I 28⊃36;
41 pred y≤z⊃tak1(x,y,z)=tak0(x,y,z)  (4 18 29)
*****⊃I 30⊃37;
42 ¬(pred y≤z)⊃tak1(x,y,z)=tak0(x,y,z)  (4 18 29)
*****TAUT tak1(x,y,z)=tak0(x,y,z) 41:42;
43 tak1(x,y,z)=tak0(x,y,z)  (4 18 29)
*****⊃I 27⊃40;
44 pred x≤y⊃tak1(x,y,z)=tak0(x,y,z)  (4 18)
*****⊃I 29⊃↑↑;
45 ¬(pred x≤y)⊃tak1(x,y,z)=tak0(x,y,z)  (4 18)
*****TAUT tak1(x,y,z)=tak0(x,y,z) 44:45;
46 tak1(x,y,z)=tak0(x,y,z)  (4 18)
*****⊃I 18⊃↑;
```

47 ¬(y≤z)⊃tak1(x,y,z)=tak0(x,y,z) (4)

*****TAUT tak1(x,y,z)=tak0(x,y,z) 17,47;

48 tak1(x,y,z)=tak0(x,y,z) (4)

*****⊃I 4⊃↑;

49 ¬(x≤y)⊃tak1(x,y,z)=tak0(x,y,z)

*****TAUT tak1(x,y,z)=tak0(x,y,z) 3,49;

50 tak1(x,y,z)=tak0(x,y,z)

7.4. The proof using GOAL.

For the GOAL proof of the Takeuchi function we used exactly the same axioms shown before. The number of user's command required by this proof is one third of the number of commands required in the previous one. On the other side, the number of line in the FOL proof generated by the GOAL unwinding mechanism is roughly the same as in the other proof.

The formulae that appear in the GOAL proof are much bigger that in McCarthy's FOL proof. But this does not seriously affect the usefulness of GOAL. In the case of this proof, I did not really have to scan much of those formulae. The commands were guessed by inspecting the main conditional of the WFF.

Our proof combined some forward proving with the GOAL commands. In total, it used nine calls to TRY and five (forward) uses of the FOL command MONADIC.

7.4.1. Comparison of the user input.

For ease of comparison, we show first the commands typed by the user in each case. The structure of the case analysis is apparent in the commands for the GOAL proof.

7.4.1.1. Commands for the forward proof.

```
ASSUME x≤y;
REWRITE tak1(x,y,z)=tak0(x,y,z) BY LOGICTREEu{ TAK0,TAK1,1};
⊃I ↑↑⊃↑;
ASSUME ¬(x≤y);
REWRITE y<x BY LOGICTREEu{ LESS9,4};
ASSUME y≤z;
ASSUME pred x≤y;
∀E LESS2 y;
∀E LESS4 pred y,y,z;
TAUT pred y≤z 6,8:9;
REWRITE tak1(x,y,z)=tak0(x,y,z) BY LOGICTREEu{ TAK0,TAK1,4:7,10};
⊃I 7⊃↑;
ASSUME ¬(pred x≤y);
REWRITE tak1(x,y,z)=tak0(x,y,z) BY LOGICTREEu{ TAK0,TAK1,LESS7,4:6,10,13};
⊃I ↑↑⊃↑;
TAUT tak1(x,y,z)=tak0(x,y,z) 12,15;
⊃I 6⊃↑;
ASSUME ¬(y≤z);
REWRITE z<y BY LOGICTREEu{ LESS9,18};
∀E LESS4 z,y,x;
∀E LESS5 z,x;
TAUT z≤x 5,19:21;
∀E LESS4 pred z,z,x;
∀E LESS2 z;
∀E LESS4 pred z,z,x;
TAUT pred z≤x 22,24:25;
ASSUME pred x≤y;
ASSUME pred y≤z;
ASSUME ¬(pred x≤y);
ASSUME ¬(pred y≤z);
REWRITE tak1(x,y,z)=tak0(x,y,z) BY FOOu{ 27:28};
REWRITE tak1(x,y,z)=tak0(x,y,z) BY FOOu{ 27,30};
∀E LESS4 pred x,z,y;
∀E LESS5 z,y;
TAUT ¬(pred x≤z) 19,29,33:34;
REWRITE tak1(x,y,z)=tak0(x,y,z) BY FOOu{ 28:29,35};
REWRITE tak1(x,y,z)=tak0(x,y,z) BY FOOu{ 29:30};
⊃I 28⊃31;
⊃I 30⊃32;
TAUT tak1(x,y,z)=tak0(x,y,z) 38:39;
⊃I 28⊃36;
⊃I 30⊃37;
TAUT tak1(x,y,z)=tak0(x,y,z) 41:42;
⊃I 27⊃40;
⊃I 29⊃↑↑;
TAUT tak1(x,y,z)=tak0(x,y,z) 44:45;
⊃I 18⊃↑;
TAUT tak1(x,y,z)=tak0(x,y,z) 17,47;
⊃I 4⊃↑;
TAUT tak1(x,y,z)=tak0(x,y,z) 3,49;
```

7.4.1.2. Commands for the goal oriented proof.

```
TRY USING ∀l;
TRY USING REWRITE BY {TAK1 TAK0};
TRY USING IFCASES x≤y;
TRY USING IFCASES y≤z;
MONADIC pred y ≤ z LESS4 LESS2 6;
MONADIC IF pred x ≤ y THEN y ELSE z ≤ z  LESS7 6;
TRY #1#1#1#2#1#1#1#1 USING REWRITE BY {↑↑,↑};
MONADIC z≤x 4 7 LESS8 LESS5 LESS4;
MONADIC pred z ≤ x LESS2 LESS4 ↑;
MONADIC IF pred y ≤ z  THEN z ELSE x  ≤ x LESS7 ↑↑;
TRY USING REWRITE BY {↑↑↑,↑↑,↑,LESS7};
TRY USING IFCASES pred y ≤ z;
TRY USING IFCASES pred x ≤ y;
TRY USING MONADIC 7 26 LESS8 LESS4;
```

7.4.2. The complete man-machine dialog.

The following is the complete protocol of the GOAL proof.

```
*****GOAL ∀x y z.(tak1(x,y,z)=tak0(x,y,z));

Goal #1:  ∀x y z.tak1(x,y,z)=tak0(x,y,z)

*****TRY USING ∀l;

Goal #1#1:  tak1(x,y,z)=tak0(x,y,z)

*****TRY USING REWRITE BY {TAK1 TAK0};

Goal #1#1#1:  IF x≤y THEN y ELSE IF IF pred x≤y THEN y ELSE IF y≤z TH%
EN z ELSE pred x≤IF pred y≤z THEN z ELSE IF z≤x THEN x ELSE pred y TH%
EN IF pred y≤z THEN z ELSE IF z≤x THEN x ELSE pred y ELSE IF IF pred %
y≤z THEN z ELSE IF z≤x THEN x ELSE pred y≤IF pred z≤x THEN x ELSE IF %
x≤y THEN y ELSE pred z THEN IF pred z≤x THEN x ELSE IF x≤y THEN y ELS%
E pred z ELSE IF pred x≤y THEN y ELSE IF y≤z THEN z ELSE pred x=IF x≤%
y THEN y ELSE IF y≤z THEN z ELSE x

*****TRY USING IFCASESHORT x≤y;

Goal #1#1#1#1:  x≤y⊃IF x≤y THEN y ELSE IF IF pred x≤y THEN y ELSE IF %
y≤z THEN z ELSE pred x≤IF pred y≤z THEN z ELSE IF z≤x THEN x ELSE pre%
d y THEN IF pred y≤z THEN z ELSE IF z≤x THEN x ELSE pred y ELSE IF IF%
```

```
 pred y≤z THEN z ELSE IF z≤x THEN x ELSE pred y≤IF pred z≤x THEN x EL%
SE IF x≤y THEN y ELSE pred z THEN IF pred z≤x THEN x ELSE IF x≤y THEN%
 y ELSE pred z ELSE IF pred x≤y THEN y ELSE IF y≤z THEN z ELSE pred x%
=IF x≤y THEN y ELSE IF y≤z THEN z ELSE x
Goal #1#1#1#2: ¬(x≤y)⊃IF x≤y THEN y ELSE IF IF pred x≤y THEN y ELSE %
IF y≤z THEN z ELSE pred x≤IF pred y≤z THEN z ELSE IF z≤x THEN x ELSE %
pred y THEN IF pred y≤z THEN z ELSE IF z≤x THEN x ELSE pred y ELSE IF%
 IF pred y≤z THEN z ELSE IF z≤x THEN x ELSE pred y≤IF pred z≤x THEN x%
 ELSE IF x≤y THEN y ELSE pred z THEN IF pred z≤x THEN x ELSE IF x≤y T%
HEN y ELSE pred z ELSE IF pred x≤y THEN y ELSE IF y≤z THEN z ELSE pre%
d x=IF x≤y THEN y ELSE IF y≤z THEN z ELSE x
Goal #1#1#1#1#1: IF x≤y THEN y ELSE IF IF pred x≤y THEN y ELSE IF y≤%
z THEN z ELSE pred x≤IF pred y≤z THEN z ELSE IF z≤x THEN x ELSE pred %
y THEN IF pred y≤z THEN z ELSE IF z≤x THEN x ELSE pred y ELSE IF IF p%
red y≤z THEN z ELSE IF z≤x THEN x ELSE pred y≤IF pred z≤x THEN x ELSE%
 IF x≤y THEN y ELSE pred z THEN IF pred z≤x THEN x ELSE IF x≤y THEN y%
 ELSE pred z ELSE IF pred x≤y THEN y ELSE IF y≤z THEN z ELSE pred x=I%
F x≤y THEN y ELSE IF y≤z THEN z ELSE x
1 x≤y  (1)

2 IF x≤y THEN y ELSE IF IF pred x≤y THEN y ELSE IF y≤z THEN z ELSE pr%
ed x≤IF pred y≤z THEN z ELSE IF z≤x THEN x ELSE pred y THEN IF pred y%
≤z THEN z ELSE IF z≤x THEN x ELSE pred y ELSE IF IF pred y≤z THEN z E%
LSE IF z≤x THEN x ELSE pred y≤IF pred z≤x THEN x ELSE IF x≤y THEN y E%
LSE pred z THEN IF pred z≤x THEN x ELSE IF x≤y THEN y ELSE pred z ELS%
E IF pred x≤y THEN y ELSE IF y≤z THEN z ELSE pred x=IF x≤y THEN y ELS%
E IF y≤z THEN z ELSE x  (1)

3 x≤y⊃IF x≤y THEN y ELSE IF IF pred x≤y THEN y ELSE IF y≤z THEN z ELS%
E pred x≤IF pred y≤z THEN z ELSE IF z≤x THEN x ELSE pred y THEN IF pr%
ed y≤z THEN z ELSE IF z≤x THEN x ELSE pred y ELSE IF IF pred y≤z THEN%
 z ELSE IF z≤x THEN x ELSE pred y≤IF pred z≤x THEN x ELSE IF x≤y THEN%
 y ELSE pred z THEN IF pred z≤x THEN x ELSE IF x≤y THEN y ELSE pred z%
 ELSE IF pred x≤y THEN y ELSE IF y≤z THEN z ELSE pred x=IF x≤y THEN y%
 ELSE IF y≤z THEN z ELSE x

Goal #1#1#1#2#1: IF x≤y THEN y ELSE IF IF pred x≤y THEN y ELSE IF y≤%
z THEN z ELSE pred x≤IF pred y≤z THEN z ELSE IF z≤x THEN x ELSE pred %
y THEN IF pred y≤z THEN z ELSE IF z≤x THEN x ELSE pred y ELSE IF IF p%
red y≤z THEN z ELSE IF z≤x THEN x ELSE pred y≤IF pred z≤x THEN x ELSE%
 IF x≤y THEN y ELSE pred z THEN IF pred z≤x THEN x ELSE IF x≤y THEN y%
 ELSE pred z ELSE IF pred x≤y THEN y ELSE IF y≤z THEN z ELSE pred x=I%
F x≤y THEN y ELSE IF y≤z THEN z ELSE x
4 ¬(x≤y)  (4)

5 x≤y=FALSE  (4)

Goal #1#1#1#2#1#1: IF IF pred x≤y THEN y ELSE IF y≤z THEN z ELSE pre%
d x≤IF pred y≤z THEN z ELSE IF z≤x THEN x ELSE pred y THEN IF pred y≤%
z THEN z ELSE IF z≤x THEN x ELSE pred y ELSE IF IF pred y≤z THEN z EL%
SE IF z≤x THEN x ELSE pred y≤IF pred z≤x THEN x ELSE pred z THEN IF p%
red z≤x THEN x ELSE pred z ELSE IF pred x≤y THEN y ELSE IF y≤z THEN z%
 ELSE pred x=IF y≤z THEN z ELSE x

*****TRY USING IFCASESHORT y≤z;
```

```
Goal #1#1#1#2#1#1#1: y≤z⊃IF IF pred x≤y THEN y ELSE IF y≤z THEN z EL%
SE pred x≤IF pred y≤z THEN z ELSE IF z≤x THEN x ELSE pred y THEN IF p%
red y≤z THEN z ELSE IF z≤x THEN x ELSE pred y ELSE IF IF pred y≤z THE%
N z ELSE IF z≤x THEN x ELSE pred y≤IF pred z≤x THEN x ELSE pred z THE%
N IF pred z≤x THEN x ELSE pred z ELSE IF pred x≤y THEN y ELSE IF y≤z %
THEN z ELSE pred x=IF y≤z THEN z ELSE x
Goal #1#1#1#2#1#1#2: ¬(y≤z)⊃IF IF pred x≤y THEN y ELSE IF y≤z THEN z%
 ELSE pred x≤IF pred y≤z THEN z ELSE IF z≤x THEN x ELSE pred y THEN I%
F pred y≤z THEN z ELSE IF z≤x THEN x ELSE pred y ELSE IF IF pred y≤z %
THEN z ELSE IF z≤x THEN x ELSE pred y≤IF pred z≤x THEN x ELSE pred z %
THEN IF pred z≤x THEN x ELSE pred z ELSE IF pred x≤y THEN y ELSE IF y%
≤z THEN z ELSE pred x=IF y≤z THEN z ELSE x
Goal #1#1#1#2#1#1#1#1: IF IF pred x≤y THEN y ELSE IF y≤z THEN z ELSE%
 pred x≤IF pred y≤z THEN z ELSE IF z≤x THEN x ELSE pred y THEN IF pre%
d y≤z THEN z ELSE IF z≤x THEN x ELSE pred y ELSE IF IF pred y≤z THEN %
z ELSE IF z≤x THEN x ELSE pred y≤IF pred z≤x THEN x ELSE pred z THEN %
IF pred z≤x THEN x ELSE pred z ELSE IF pred x≤y THEN y ELSE IF y≤z TH%
EN z ELSE pred x=IF y≤z THEN z ELSE x
6 y≤z  (6)

Goal #1#1#1#2#1#1#1#1#1: IF IF pred x≤y THEN y ELSE z≤IF pred y≤z TH%
EN z ELSE IF z≤x THEN x ELSE pred y THEN IF pred y≤z THEN z ELSE IF z%
≤x THEN x ELSE pred y ELSE IF IF pred y≤z THEN z ELSE IF z≤x THEN x E%
LSE pred y≤IF pred z≤x THEN x ELSE pred z THEN IF pred z≤x THEN x ELS%
E pred z ELSE IF pred x≤y THEN y ELSE z=z
Goal #1#1#1#2#1#1#2#1: IF IF pred x≤y THEN y ELSE IF y≤z THEN z ELSE%
 pred x≤IF pred y≤z THEN z ELSE IF z≤x THEN x ELSE pred y THEN IF pre%
d y≤z THEN z ELSE IF z≤x THEN x ELSE pred y ELSE IF IF pred y≤z THEN %
z ELSE IF z≤x THEN x ELSE pred y≤IF pred z≤x THEN x ELSE pred z THEN %
IF pred z≤x THEN x ELSE pred z ELSE IF pred x≤y THEN y ELSE IF y≤z TH%
EN z ELSE pred x=IF y≤z THEN z ELSE x
7 ¬(y≤z)  (7)

8 y≤z≡FALSE  (7)

Goal #1#1#1#2#1#1#2#1#1: IF IF pred x≤y THEN y ELSE pred x≤IF pred y%
≤z THEN z ELSE IF z≤x THEN x ELSE pred y THEN IF pred y≤z THEN z ELSE%
 IF z≤x THEN x ELSE pred y ELSE IF IF pred y≤z THEN z ELSE IF z≤x THE%
N x ELSE pred y≤IF pred z≤x THEN x ELSE pred z THEN IF pred z≤x THEN %
x ELSE pred z ELSE IF pred x≤y THEN y ELSE pred x=x

*****MONADIC pred y ≤ z LESS4 LESS2 6;

9 pred y≤z  (6)

*****MONADIC IF pred x ≤ y THEN y ELSE z ≤ z  LESS7 6;

10 IF pred x≤y THEN y ELSE z≤z  (6)

*****TRY #1#1#1#2#1#1#1#1#1 USING REWRITE BY {↑↑,↑};

11 IF IF pred x≤y THEN y ELSE z≤IF pred y≤z THEN z ELSE IF z≤x THEN x%
 ELSE pred y THEN IF pred y≤z THEN z ELSE IF z≤x THEN x ELSE pred y E%
LSE IF IF pred y≤z THEN z ELSE IF z≤x THEN x ELSE pred y≤IF pred z≤x %
THEN x ELSE pred z THEN IF pred z≤x THEN x ELSE pred z ELSE IF pred x%
```

```
≤y THEN y ELSE z=z  (6)

12 IF IF pred x≤y THEN y ELSE IF y≤z THEN z ELSE pred x≤IF pred y≤z T%
HEN z ELSE IF z≤x THEN x ELSE pred y THEN IF pred y≤z THEN z ELSE IF %
z≤x THEN x ELSE pred y ELSE IF IF pred y≤z THEN z ELSE IF z≤x THEN x %
ELSE pred y≤IF pred z≤x THEN x ELSE pred z THEN IF pred z≤x THEN x EL%
SE pred z ELSE IF pred x≤y THEN y ELSE IF y≤z THEN z ELSE pred x=IF y%
≤z THEN z ELSE x=IF IF pred x≤y THEN y ELSE z≤IF pred y≤z THEN z ELSE%
 IF z≤x THEN x ELSE pred y THEN IF pred y≤z THEN z ELSE IF z≤x THEN x%
 ELSE pred y ELSE IF IF pred y≤z THEN z ELSE IF z≤x THEN x ELSE pred %
y≤IF pred z≤x THEN x ELSE pred z THEN IF pred z≤x THEN x ELSE pred z %
ELSE IF pred x≤y THEN y ELSE z=z  (6)

13 IF IF pred x≤y THEN y ELSE IF y≤z THEN z ELSE pred x≤IF pred y≤z T%
HEN z ELSE IF z≤x THEN x ELSE pred y THEN IF pred y≤z THEN z ELSE IF %
z≤x THEN x ELSE pred y ELSE IF IF pred y≤z THEN z ELSE IF z≤x THEN x %
ELSE pred y≤IF pred z≤x THEN x ELSE pred z THEN IF pred z≤x THEN x EL%
SE pred z ELSE IF pred x≤y THEN y ELSE IF y≤z THEN z ELSE pred x=IF y%
≤z THEN z ELSE x  (6)

14 y≤z⊃IF IF pred x≤y THEN y ELSE IF y≤z THEN z ELSE pred x≤IF pred y%
≤z THEN z ELSE IF z≤x THEN x ELSE pred y THEN IF pred y≤z THEN z ELSE%
 IF z≤x THEN x ELSE pred y ELSE IF IF pred y≤z THEN z ELSE IF z≤x THE%
N x ELSE pred y≤IF pred z≤x THEN x ELSE pred z THEN IF pred z≤x THEN %
x ELSE pred z ELSE IF pred x≤y THEN y ELSE IF y≤z THEN z ELSE pred x=%
IF y≤z THEN z ELSE x

*****MONADIC z≤x 4 7 LESS8 LESS5 LESS4;

15 z≤x  (4 7)

*****MONADIC pred z ≤ x LESS2 LESS4 ↑;

16 pred z≤x  (4 7)

*****MONADIC IF pred y ≤ z  THEN z ELSE x  ≤ x LESS7 ↑↑;

17 IF pred y≤z THEN z ELSE x≤x  (4 7)

*****TRY USING REWRITE BY {↑↑↑,↑↑,↑,LESS7};

Goal *1*1*1*2*1*1*2*1*1*1: IF IF pred x≤y THEN y ELSE pred x≤IF pred%
 y≤z THEN z ELSE x THEN IF pred y≤z THEN z ELSE x ELSE x=x

*****TRY USING IFCASESHORT pred y ≤ z;

Goal *1*1*1*2*1*1*2*1*1*1*1: pred y≤z⊃IF IF pred x≤y THEN y ELSE pre%
d x≤IF pred y≤z THEN z ELSE x THEN IF pred y≤z THEN z ELSE x ELSE x=x
Goal *1*1*1*2*1*1*2*1*1*1*2: ¬(pred y≤z)⊃IF IF pred x≤y THEN y ELSE %
pred x≤IF pred y≤z THEN z ELSE x THEN IF pred y≤z THEN z ELSE x ELSE %
x=x
Goal *1*1*1*2*1*1*2*1*1*1*1*1: IF IF pred x≤y THEN y ELSE pred x≤IF %
pred y≤z THEN z ELSE x THEN IF pred y≤z THEN z ELSE x ELSE x=x
18 pred y≤z  (18)
```

```
Goal #1#1#1#2#1#1#2#1#1#1#1#1#1: IF IF pred x≤y THEN y ELSE pred x≤z%
 THEN z ELSE x=x
Goal #1#1#1#2#1#1#2#1#1#1#2#1: IF IF pred x≤y THEN y ELSE pred x≤IF %
pred y≤z THEN z ELSE x THEN IF pred y≤z THEN z ELSE x ELSE x=x
19 ¬(pred y≤z) (19)

20 pred y≤z≡FALSE (19)

21 IF IF pred x≤y THEN y ELSE pred x≤IF pred y≤z THEN z ELSE x THEN I%
F pred y≤z THEN z ELSE x ELSE x=x (19)

22 ¬(pred y≤z)⊃IF IF pred x≤y THEN y ELSE pred x≤IF pred y≤z THEN z E%
LSE x THEN IF pred y≤z THEN z ELSE x ELSE x=x

*****TRY USING IFCASESHORT pred x ≤ y;

Goal #1#1#1#2#1#1#2#1#1#1#1#1#1: pred x≤y⊃IF IF pred x≤y THEN y EL%
SE pred x≤z THEN z ELSE x=x
Goal #1#1#1#2#1#1#2#1#1#1#1#1#2: ¬(pred x≤y)⊃IF IF pred x≤y THEN y%
 ELSE pred x≤z THEN z ELSE x=x
Goal #1#1#1#2#1#1#2#1#1#1#1#1#1: IF IF pred x≤y THEN y ELSE pred%
 x≤z THEN z ELSE x=x
23 pred x≤y (23)

24 IF IF pred x≤y THEN y ELSE pred x≤z THEN z ELSE x=x (7 23)

25 pred x≤y⊃IF IF pred x≤y THEN y ELSE pred x≤z THEN z ELSE x=x (7)

Goal #1#1#1#2#1#1#2#1#1#1#1#1#2#1: IF IF pred x≤y THEN y ELSE pred%
 x≤z THEN z ELSE x=x
26 ¬(pred x≤y) (26)

27 pred x≤y≡FALSE (26)

Goal #1#1#1#2#1#1#2#1#1#1#1#1#2#1#1: IF pred x≤z THEN z ELSE x=x

*****TRY USING MONADIC 7 26 LESS8 LESS4;

28 IF pred x≤z THEN z ELSE x=x (7 26)

29 IF IF pred x≤y THEN y ELSE pred x≤z THEN z ELSE x=x≡IF pred x≤z TH%
EN z ELSE x=x (26)

30 IF IF pred x≤y THEN y ELSE pred x≤z THEN z ELSE x=x (7 26)

31 ¬(pred x≤y)⊃IF IF pred x≤y THEN y ELSE pred x≤z THEN z ELSE x=x (%
7)

32 IF IF pred x≤y THEN y ELSE pred x≤z THEN z ELSE x=x (7)

33 IF IF pred x≤y THEN y ELSE pred x≤IF pred y≤z THEN z ELSE x THEN I%
F pred y≤z THEN z ELSE x ELSE x=x≡IF IF pred x≤y THEN y ELSE pred x≤z%
 THEN z ELSE x=x (18)
```

```
34 IF IF pred x≤y THEN y ELSE pred x≤IF pred y≤z THEN z ELSE x THEN I%
F pred y≤z THEN z ELSE x ELSE x=x  (7 18)

35 pred y≤z⊃IF IF pred x≤y THEN y ELSE pred x≤IF pred y≤z THEN z ELSE%
 x THEN IF pred y≤z THEN z ELSE x ELSE x=x  (7)

36 IF IF pred x≤y THEN y ELSE pred x≤IF pred y≤z THEN z ELSE x THEN I%
F pred y≤z THEN z ELSE x ELSE x=x  (7)

37 IF IF pred x≤y THEN y ELSE pred x≤IF pred y≤z THEN z ELSE IF z≤x T%
HEN x ELSE pred y THEN IF pred y≤z THEN z ELSE IF z≤x THEN x ELSE pre%
d y ELSE IF IF pred y≤z THEN z ELSE IF z≤x THEN x ELSE pred y≤IF pred%
 z≤x THEN x ELSE pred z THEN IF pred z≤x THEN x ELSE pred z ELSE IF p%
red x≤y THEN y ELSE pred x=x≡IF IF pred x≤y THEN y ELSE pred x≤IF pre%
d y≤z THEN z ELSE x THEN IF pred y≤z THEN z ELSE x ELSE x=x  (4 7)

38 IF IF pred x≤y THEN y ELSE pred x≤IF pred y≤z THEN z ELSE IF z≤x T%
HEN x ELSE pred y THEN IF pred y≤z THEN z ELSE IF z≤x THEN x ELSE pre%
d y ELSE IF IF pred y≤z THEN z ELSE IF z≤x THEN x ELSE pred y≤IF pred%
 z≤x THEN x ELSE pred z THEN IF pred z≤x THEN x ELSE pred z ELSE IF p%
red x≤y THEN y ELSE pred x=x  (4 7)

39 IF IF pred x≤y THEN y ELSE IF y≤z THEN z ELSE pred x≤IF pred y≤z T%
HEN z ELSE IF z≤x THEN x ELSE pred y THEN IF pred y≤z THEN z ELSE IF %
z≤x THEN x ELSE pred y ELSE IF IF pred y≤z THEN z ELSE IF z≤x THEN x %
ELSE pred y≤IF pred z≤x THEN x ELSE pred z THEN IF pred z≤x THEN x EL%
SE pred z ELSE IF pred x≤y THEN y ELSE IF y≤z THEN z ELSE pred x=IF y%
≤z THEN z ELSE x≡IF IF pred x≤y THEN y ELSE pred x≤IF pred y≤z THEN z%
 ELSE IF z≤x THEN x ELSE pred y THEN IF pred y≤z THEN z ELSE IF z≤x T%
HEN x ELSE pred y ELSE IF IF pred y≤z THEN z ELSE IF z≤x THEN x ELSE %
pred y≤IF pred z≤x THEN x ELSE pred z THEN IF pred z≤x THEN x ELSE pr%
ed z ELSE IF pred x≤y THEN y ELSE pred x=x  (7)

40 IF IF pred x≤y THEN y ELSE IF y≤z THEN z ELSE pred x≤IF pred y≤z T%
HEN z ELSE IF z≤x THEN x ELSE pred y THEN IF pred y≤z THEN z ELSE IF %
z≤x THEN x ELSE pred y ELSE IF IF pred y≤z THEN z ELSE IF z≤x THEN x %
ELSE pred y≤IF pred z≤x THEN x ELSE pred z THEN IF pred z≤x THEN x EL%
SE pred z ELSE IF pred x≤y THEN y ELSE IF y≤z THEN z ELSE pred x=IF y%
≤z THEN z ELSE x  (4 7)

41 ¬(y≤z)⊃IF IF pred x≤y THEN y ELSE IF y≤z THEN z ELSE pred x≤IF pre%
d y≤z THEN z ELSE IF z≤x THEN x ELSE pred y THEN IF pred y≤z THEN z E%
LSE IF z≤x THEN x ELSE pred y ELSE IF IF pred y≤z THEN z ELSE IF z≤x %
THEN x ELSE pred y≤IF pred z≤x THEN x ELSE pred z THEN IF pred z≤x TH%
EN x ELSE pred z ELSE IF pred x≤y THEN y ELSE IF y≤z THEN z ELSE pred%
 x=IF y≤z THEN z ELSE x  (4)

42 IF IF pred x≤y THEN y ELSE IF y≤z THEN z ELSE pred x≤IF pred y≤z T%
HEN z ELSE IF z≤x THEN x ELSE pred y THEN IF pred y≤z THEN z ELSE IF %
z≤x THEN x ELSE pred y ELSE IF IF pred y≤z THEN z ELSE IF z≤x THEN x %
ELSE pred y≤IF pred z≤x THEN x ELSE pred z THEN IF pred z≤x THEN x EL%
SE pred z ELSE IF pred x≤y THEN y ELSE IF y≤z THEN z ELSE pred x=IF y%
≤z THEN z ELSE x  (4)

43 IF x≤y THEN y ELSE IF IF pred x≤y THEN y ELSE IF y≤z THEN z ELSE p%
```

```
red x≤IF pred y≤z THEN z ELSE IF z≤x THEN x ELSE pred y THEN IF pred %
y≤z THEN z ELSE IF z≤x THEN x ELSE pred y ELSE IF IF pred y≤z THEN z %
ELSE IF z≤x THEN x ELSE pred y≤IF pred z≤x THEN x ELSE IF x≤y THEN y %
ELSE pred z THEN IF pred z≤x THEN x ELSE IF x≤y THEN y ELSE pred z EL%
SE IF pred x≤y THEN y ELSE IF y≤z THEN z ELSE pred x=IF x≤y THEN y EL%
SE IF y≤z THEN z ELSE x=IF IF pred x≤y THEN y ELSE IF y≤z THEN z ELSE%
 pred x≤IF pred y≤z THEN z ELSE IF z≤x THEN x ELSE pred y THEN IF pre%
d y≤z THEN z ELSE IF z≤x THEN x ELSE pred y ELSE IF IF pred y≤z THEN %
z ELSE IF z≤x THEN x ELSE pred y≤IF pred z≤x THEN x ELSE pred z THEN %
IF pred z≤x THEN x ELSE pred z ELSE IF pred x≤y THEN y ELSE IF y≤z TH%
EN z ELSE pred x=IF y≤z THEN z ELSE x  (4)

44 IF x≤y THEN y ELSE IF IF pred x≤y THEN y ELSE IF y≤z THEN z ELSE p%
red x≤IF pred y≤z THEN z ELSE IF z≤x THEN x ELSE pred y THEN IF pred %
y≤z THEN z ELSE IF z≤x THEN x ELSE pred y ELSE IF IF pred y≤z THEN z %
ELSE IF z≤x THEN x ELSE pred y≤IF pred z≤x THEN x ELSE IF x≤y THEN y %
ELSE pred z THEN IF pred z≤x THEN x ELSE IF x≤y THEN y ELSE pred z EL%
SE IF pred x≤y THEN y ELSE IF y≤z THEN z ELSE pred x=IF x≤y THEN y EL%
SE IF y≤z THEN z ELSE x  (4)

45 ¬(x≤y)⊃IF x≤y THEN y ELSE IF IF pred x≤y THEN y ELSE IF y≤z THEN z%
 ELSE pred x≤IF pred y≤z THEN z ELSE IF z≤x THEN x ELSE pred y THEN I%
F pred y≤z THEN z ELSE IF z≤x THEN x ELSE pred y ELSE IF IF pred y≤z %
THEN z ELSE IF z≤x THEN x ELSE pred y≤IF pred z≤x THEN x ELSE IF x≤y %
THEN y ELSE pred z THEN IF pred z≤x THEN x ELSE IF x≤y THEN y ELSE pr%
ed z ELSE IF pred x≤y THEN y ELSE IF y≤z THEN z ELSE pred x=IF x≤y TH%
EN y ELSE IF y≤z THEN z ELSE x

46 IF x≤y THEN y ELSE IF IF pred x≤y THEN y ELSE IF y≤z THEN z ELSE p%
red x≤IF pred y≤z THEN z ELSE IF z≤x THEN x ELSE pred y THEN IF pred %
y≤z THEN z ELSE IF z≤x THEN x ELSE pred y ELSE IF IF pred y≤z THEN z %
ELSE IF z≤x THEN x ELSE pred y≤IF pred z≤x THEN x ELSE IF x≤y THEN y %
ELSE pred z THEN IF pred z≤x THEN x ELSE IF x≤y THEN y ELSE pred z EL%
SE IF pred x≤y THEN y ELSE IF y≤z THEN z ELSE pred x=IF x≤y THEN y EL%
SE IF y≤z THEN z ELSE x

47 tak1(x,y,z)=tak0(x,y,z)=IF x≤y THEN y ELSE IF IF pred x≤y THEN y E%
LSE IF y≤z THEN z ELSE pred x≤IF pred y≤z THEN z ELSE IF z≤x THEN x E%
LSE pred y THEN IF pred y≤z THEN z ELSE IF z≤x THEN x ELSE pred y ELS%
E IF IF pred y≤z THEN z ELSE IF z≤x THEN x ELSE pred y≤IF pred z≤x TH%
EN x ELSE IF x≤y THEN y ELSE pred z THEN IF pred z≤x THEN x ELSE IF x%
≤y THEN y ELSE pred z ELSE IF pred x≤y THEN y ELSE IF y≤z THEN z ELSE%
 pred x=IF x≤y THEN y ELSE IF y≤z THEN z ELSE x

48 tak1(x,y,z)=tak0(x,y,z)

49 ∀x y z.tak1(x,y,z)=tak0(x,y,z)
```

7.4.3. The FOL proof generated by GOAL.

We have stressed the fact that GOAL always generates a FOL proof that is indistinguishable from a user generated proof. For the sake of completeness, we also show here the FOL proof that results from the previous dialog.

```
*****SHOW PROOF;

*****ASSUME x≤y;

1 x≤y  (1)

*****REWRITE IF x≤y THEN y ELSE IF IF pred x≤y THEN y ELSE IF y≤z THE%
N z ELSE pred x≤IF pred y≤z THEN z ELSE IF z≤x THEN x ELSE pred y THE%
N IF pred y≤z THEN z ELSE IF z≤x THEN x ELSE pred y ELSE IF IF pred y%
≤z THEN z ELSE IF z≤x THEN x ELSE pred y≤IF pred z≤x THEN x ELSE IF x%
≤y THEN y ELSE pred z THEN IF pred z≤x THEN x ELSE IF x≤y THEN y ELSE%
 pred z ELSE IF pred x≤y THEN y ELSE IF y≤z THEN z ELSE pred x=IF x≤y%
 THEN y ELSE IF y≤z THEN z ELSE x  BY  LOGICTREE COMPTREE ↑;

2 IF x≤y THEN y ELSE IF IF pred x≤y THEN y ELSE IF y≤z THEN z ELSE pr%
ed x≤IF pred y≤z THEN z ELSE IF z≤x THEN x ELSE pred y THEN IF pred y%
≤z THEN z ELSE IF z≤x THEN x ELSE pred y ELSE IF IF pred y≤z THEN z E%
LSE IF z≤x THEN x ELSE pred y≤IF pred z≤x THEN x ELSE IF x≤y THEN y E%
LSE pred z THEN IF pred z≤x THEN x ELSE IF x≤y THEN y ELSE pred z ELS%
E IF pred x≤y THEN y ELSE IF y≤z THEN z ELSE pred x=IF x≤y THEN y ELS%
E IF y≤z THEN z ELSE x  (1)

*****⊃I ↑↑⊃↑;

3 x≤y⊃IF x≤y THEN y ELSE IF IF pred x≤y THEN y ELSE IF y≤z THEN z ELS%
E pred x≤IF pred y≤z THEN z ELSE IF z≤x THEN x ELSE pred y THEN IF pr%
ed y≤z THEN z ELSE IF z≤x THEN x ELSE pred y ELSE IF IF pred y≤z THEN%
 z ELSE IF z≤x THEN x ELSE pred y≤IF pred z≤x THEN x ELSE IF x≤y THEN%
 y ELSE pred z THEN IF pred z≤x THEN x ELSE IF x≤y THEN y ELSE pred z%
 ELSE IF pred x≤y THEN y ELSE IF y≤z THEN z ELSE pred x=IF x≤y THEN y%
 ELSE IF y≤z THEN z ELSE x

*****ASSUME ¬(x≤y);

4 ¬(x≤y)  (4)

*****REWRITE ↑ BY LOGICTREE;

5 x≤y=FALSE  (4)

*****ASSUME y≤z;

6 y≤z  (6)
```

```
*****ASSUME ¬(y≤z);

7 ¬(y≤z)  (7)

*****REWRITE ↑ BY LOGICTREE;

8 y≤z=FALSE  (7)

*****;

9 pred y≤z  (6)

*****;

10 IF pred x≤y THEN y ELSE z≤z  (6)

*****REWRITE IF IF pred x≤y THEN y ELSE z≤IF pred y≤z THEN z ELSE IF %
z≤x THEN x ELSE pred y THEN IF pred y≤z THEN z ELSE IF z≤x THEN x ELS%
E pred y ELSE IF IF pred y≤z THEN z ELSE IF z≤x THEN x ELSE pred y≤IF%
 pred z≤x THEN x ELSE pred z THEN IF pred z≤x THEN x ELSE pred z ELSE%
 IF pred x≤y THEN y ELSE z=z  BY  5 LOGICTREE COMPTREE BY { 9:10};

11 IF IF pred x≤y THEN y ELSE z≤IF pred y≤z THEN z ELSE IF z≤x THEN x%
 ELSE pred y THEN IF pred y≤z THEN z ELSE IF z≤x THEN x ELSE pred y E%
LSE IF IF pred y≤z THEN z ELSE IF z≤x THEN x ELSE pred y≤IF pred z≤x %
THEN x ELSE pred z THEN IF pred z≤x THEN x ELSE pred z ELSE IF pred x%
≤y THEN y ELSE z=z  (6)

*****REWRITE IF IF pred x≤y THEN y ELSE IF y≤z THEN z ELSE pred x≤IF %
pred y≤z THEN z ELSE IF z≤x THEN x ELSE pred y THEN IF pred y≤z THEN %
z ELSE IF z≤x THEN x ELSE pred y ELSE IF IF pred y≤z THEN z ELSE IF z%
≤x THEN x ELSE pred y≤IF pred z≤x THEN x ELSE pred z THEN IF pred z≤x%
 THEN x ELSE pred z ELSE IF pred x≤y THEN y ELSE IF y≤z THEN z ELSE p%
red x=IF y≤z THEN z ELSE x  BY  5 LOGICTREE COMPTREE6;

12 IF IF pred x≤y THEN y ELSE IF y≤z THEN z ELSE pred x≤IF pred y≤z T%
HEN z ELSE IF z≤x THEN x ELSE pred y THEN IF pred y≤z THEN z ELSE IF %
z≤x THEN x ELSE pred y ELSE IF IF pred y≤z THEN z ELSE IF z≤x THEN x %
ELSE pred y≤IF pred z≤x THEN x ELSE pred z THEN IF pred z≤x THEN x EL%
SE pred z ELSE IF pred x≤y THEN y ELSE IF y≤z THEN z ELSE pred x=IF y%
≤z THEN z ELSE x=IF IF pred x≤y THEN y ELSE z≤IF pred y≤z THEN z ELSE%
 IF z≤x THEN x ELSE pred y THEN IF pred y≤z THEN z ELSE IF z≤x THEN x%
 ELSE pred y ELSE IF IF pred y≤z THEN z ELSE IF z≤x THEN x ELSE pred %
y≤IF pred z≤x THEN x ELSE pred z THEN IF pred z≤x THEN x ELSE pred z %
ELSE IF pred x≤y THEN y ELSE z=z  (6)

*****TAUT IF IF pred x≤y THEN y ELSE IF y≤z THEN z ELSE pred x≤IF pre%
d y≤z THEN z ELSE IF z≤x THEN x ELSE pred y THEN IF pred y≤z THEN z E%
LSE IF z≤x THEN x ELSE pred y ELSE IF IF pred y≤z THEN z ELSE IF z≤x %
THEN x ELSE pred y≤IF pred z≤x THEN x ELSE pred z THEN IF pred z≤x TH%
EN x ELSE pred z ELSE IF pred x≤y THEN y ELSE IF y≤z THEN z ELSE pred%
 x=IF y≤z THEN z ELSE x 11,12;

13 IF IF pred x≤y THEN y ELSE IF y≤z THEN z ELSE pred x≤IF pred y≤z T%
```

```
HEN z ELSE IF z≤x THEN x ELSE pred y THEN IF pred y≤z THEN z ELSE IF %
z≤x THEN x ELSE pred y ELSE IF IF pred y≤z THEN z ELSE IF z≤x THEN x %
ELSE pred y≤IF pred z≤x THEN x ELSE pred z THEN IF pred z≤x THEN x EL%
SE pred z ELSE IF pred x≤y THEN y ELSE IF y≤z THEN z ELSE pred x=IF y%
≤z THEN z ELSE x  (6)

*****⊃| 6⊃↑;

14 y≤z⊃IF IF pred x≤y THEN y ELSE IF y≤z THEN z ELSE pred x≤IF pred y%
≤z THEN z ELSE IF z≤x THEN x ELSE pred y THEN IF pred y≤z THEN z ELSE%
 IF z≤x THEN x ELSE pred y ELSE IF IF pred y≤z THEN z ELSE IF z≤x THE%
N x ELSE pred y≤IF pred z≤x THEN x ELSE pred z THEN IF pred z≤x THEN %
x ELSE pred z ELSE IF pred x≤y THEN y ELSE IF y≤z THEN z ELSE pred x=%
IF y≤z THEN z ELSE x

*****;

15 z≤x  (4 7)

*****;

16 pred z≤x  (4 7)

*****;

17 IF pred y≤z THEN z ELSE x≤x  (4 7)

*****ASSUME pred y≤z;

18 pred y≤z  (18)

*****ASSUME ¬(pred y≤z);

19 ¬(pred y≤z)  (19)

*****REWRITE ↑ BY LOGICTREE;

20 pred y≤z=FALSE  (19)

*****REWRITE IF IF pred x≤y THEN y ELSE pred x≤IF pred y≤z THEN z ELS%
E x THEN IF pred y≤z THEN z ELSE x ELSE x=x  BY  ↑ 8 5 LOGICTREE COMP%
TREE ↑;

21 IF IF pred x≤y THEN y ELSE pred x≤IF pred y≤z THEN z ELSE x THEN I%
F pred y≤z THEN z ELSE x ELSE x=x  (19)

*****⊃| ↑↑↑⊃↑;

22 ¬(pred y≤z)⊃IF IF pred x≤y THEN y ELSE pred x≤IF pred y≤z THEN z E%
LSE x THEN IF pred y≤z THEN z ELSE x ELSE x=x

*****ASSUME pred x≤y;

23 pred x≤y  (23)
```

```
*****REWRITE IF IF pred x≤y THEN y ELSE pred x≤z THEN z ELSE x=x  BY %
 8 5 LOGICTREE COMPTREE ↑;

24 IF IF pred x≤y THEN y ELSE pred x≤z THEN z ELSE x=x  (7 23)

*****⊃I ↑↑⊃↑;

25 pred x≤y⊃IF IF pred x≤y THEN y ELSE pred x≤z THEN z ELSE x=x  (7)

*****ASSUME ¬(pred x≤y);

26 ¬(pred x≤y)  (26)

*****REWRITE ↑ BY LOGICTREE;

27 pred x≤y=FALSE  (26)

*****MONADIC LESS4LESS8 ↑↑ 7;

28 IF pred x≤z THEN z ELSE x=x  (7 26)

*****REWRITE IF IF pred x≤y THEN y ELSE pred x≤z THEN z ELSE x=x  BY %
 ↑↑ 8 5 LOGICTREE COMPTREE ↑↑;

29 IF IF pred x≤y THEN y ELSE pred x≤z THEN z ELSE x=x=IF pred x≤z TH%
EN z ELSE x=x  (26)

*****TAUT IF IF pred x≤y THEN y ELSE pred x≤z THEN z ELSE x=x 28,29;

30 IF IF pred x≤y THEN y ELSE pred x≤z THEN z ELSE x=x  (7 26)

*****⊃I 26⊃↑;

31 ¬(pred x≤y)⊃IF IF pred x≤y THEN y ELSE pred x≤z THEN z ELSE x=x  (%
7)

*****TAUTEQ IF IF pred x≤y THEN y ELSE pred x≤z THEN z ELSE x=x 25,31%
;

32 IF IF pred x≤y THEN y ELSE pred x≤z THEN z ELSE x=x  (7)

*****REWRITE IF IF pred x≤y THEN y ELSE pred x≤IF pred y≤z THEN z ELS%
E x THEN IF pred y≤z THEN z ELSE x ELSE x=x  BY  8 5 LOGICTREE COMPTR%
EE18;

33 IF IF pred x≤y THEN y ELSE pred x≤IF pred y≤z THEN z ELSE x THEN I%
F pred y≤z THEN z ELSE x ELSE x=x=IF IF pred x≤y THEN y ELSE pred x≤z%
 THEN z ELSE x=x  (18)

*****TAUT IF IF pred x≤y THEN y ELSE pred x≤IF pred y≤z THEN z ELSE x%
 THEN IF pred y≤z THEN z ELSE x ELSE x=x 32,33;

34 IF IF pred x≤y THEN y ELSE pred x≤IF pred y≤z THEN z ELSE x THEN I%
```

```
F pred y≤z THEN z ELSE x ELSE x=x  (7 18)

*****⊃I 18⊃↑;

35 pred y≤z⊃IF IF pred x≤y THEN y ELSE pred x≤IF pred y≤z THEN z ELSE%
 x THEN IF pred y≤z THEN z ELSE x ELSE x=x  (7)

*****TAUTEQ IF IF pred x≤y THEN y ELSE pred x≤IF pred y≤z THEN z ELSE%
 x THEN IF pred y≤z THEN z ELSE x ELSE x=x 22,35;

36 IF IF pred x≤y THEN y ELSE pred x≤IF pred y≤z THEN z ELSE x THEN I%
F pred y≤z THEN z ELSE x ELSE x=x  (7)

*****REWRITE IF IF pred x≤y THEN y ELSE pred x≤IF pred y≤z THEN z ELS%
E IF z≤x THEN x ELSE pred y THEN IF pred y≤z THEN z ELSE IF z≤x THEN %
x ELSE pred y ELSE IF IF pred y≤z THEN z ELSE IF z≤x THEN x ELSE pred%
 y≤IF pred z≤x THEN x ELSE pred z THEN IF pred z≤x THEN x ELSE pred z%
 ELSE IF pred x≤y THEN y ELSE pred x=x  BY  8 5 LOGICTREE COMPTREE BY%
 { LESS7,15:17};

37 IF IF pred x≤y THEN y ELSE pred x≤IF pred y≤z THEN z ELSE IF z≤x T%
HEN x ELSE pred y THEN IF pred y≤z THEN z ELSE IF z≤x THEN x ELSE pre%
d y ELSE IF IF pred y≤z THEN z ELSE IF z≤x THEN x ELSE pred y≤IF pred%
 z≤x THEN x ELSE pred z THEN IF pred z≤x THEN x ELSE pred z ELSE IF p%
red x≤y THEN y ELSE pred x=x≡IF IF pred x≤y THEN y ELSE pred x≤IF pre%
d y≤z THEN z ELSE x THEN IF pred y≤z THEN z ELSE x ELSE x=x  (4 7)

*****TAUT IF IF pred x≤y THEN y ELSE pred x≤IF pred y≤z THEN z ELSE I%
F z≤x THEN x ELSE pred y THEN IF pred y≤z THEN z ELSE IF z≤x THEN x E%
LSE pred y ELSE IF IF pred y≤z THEN z ELSE IF z≤x THEN x ELSE pred y≤%
IF pred z≤x THEN x ELSE pred z THEN IF pred z≤x THEN x ELSE pred z EL%
SE IF pred x≤y THEN y ELSE pred x=x 36,37;

38 IF IF pred x≤y THEN y ELSE pred x≤IF pred y≤z THEN z ELSE IF z≤x T%
HEN x ELSE pred y THEN IF pred y≤z THEN z ELSE IF z≤x THEN x ELSE pre%
d y ELSE IF IF pred y≤z THEN z ELSE IF z≤x THEN x ELSE pred y≤IF pred%
 z≤x THEN x ELSE pred z THEN IF pred z≤x THEN x ELSE pred z ELSE IF p%
red x≤y THEN y ELSE pred x=x  (4 7)

*****REWRITE IF IF pred x≤y THEN y ELSE IF y≤z THEN z ELSE pred x≤IF %
pred y≤z THEN z ELSE IF z≤x THEN x ELSE pred y THEN IF pred y≤z THEN %
z ELSE IF z≤x THEN x ELSE pred y ELSE IF IF pred y≤z THEN z ELSE IF z%
≤x THEN x ELSE pred y≤IF pred z≤x THEN x ELSE pred z THEN IF pred z≤x%
 THEN x ELSE pred z ELSE IF pred x≤y THEN y ELSE IF y≤z THEN z ELSE p%
red x=IF y≤z THEN z ELSE x  BY  8 5 LOGICTREE COMPTREE8;

39 IF IF pred x≤y THEN y ELSE IF y≤z THEN z ELSE pred x≤IF pred y≤z T%
HEN z ELSE IF z≤x THEN x ELSE pred y THEN IF pred y≤z THEN z ELSE IF %
z≤x THEN x ELSE pred y ELSE IF IF pred y≤z THEN z ELSE IF z≤x THEN x %
ELSE pred y≤IF pred z≤x THEN x ELSE pred z THEN IF pred z≤x THEN x EL%
SE pred z ELSE IF pred x≤y THEN y ELSE IF y≤z THEN z ELSE pred x=IF y%
≤z THEN z ELSE x≡IF IF pred x≤y THEN y ELSE pred x≤IF pred y≤z THEN z%
 ELSE IF z≤x THEN x ELSE pred y THEN IF pred y≤z THEN z ELSE IF z≤x T%
HEN x ELSE pred y ELSE IF IF pred y≤z THEN z ELSE IF z≤x THEN x ELSE %
pred y≤IF pred z≤x THEN x ELSE pred z THEN IF pred z≤x THEN x ELSE pr%
```

```
ed z ELSE IF pred x≤y THEN y ELSE pred x=x  (7)

*****TAUT IF IF pred x≤y THEN y ELSE IF y≤z THEN z ELSE pred x≤IF pre%
d y≤z THEN z ELSE IF z≤x THEN x ELSE pred y THEN IF pred y≤z THEN z E%
LSE IF z≤x THEN x ELSE pred y ELSE IF IF pred y≤z THEN z ELSE IF z≤x %
THEN x ELSE pred y≤IF pred z≤x THEN x ELSE pred z THEN IF pred z≤x TH%
EN x ELSE pred z ELSE IF pred x≤y THEN y ELSE IF y≤z THEN z ELSE pred%
 x=IF y≤z THEN z ELSE x 38,39;

40 IF IF pred x≤y THEN y ELSE IF y≤z THEN z ELSE pred x≤IF pred y≤z T%
HEN z ELSE IF z≤x THEN x ELSE pred y THEN IF pred y≤z THEN z ELSE IF %
z≤x THEN x ELSE pred y ELSE IF IF pred y≤z THEN z ELSE IF z≤x THEN x %
ELSE pred y≤IF pred z≤x THEN x ELSE pred z THEN IF pred z≤x THEN x EL%
SE pred z ELSE IF pred x≤y THEN y ELSE IF y≤z THEN z ELSE pred x=IF y%
≤z THEN z ELSE x  (4 7)

*****⊃I 7⊃↑;

41 ¬(y≤z)⊃IF IF pred x≤y THEN y ELSE IF y≤z THEN z ELSE pred x≤IF pre%
d y≤z THEN z ELSE IF z≤x THEN x ELSE pred y THEN IF pred y≤z THEN z E%
LSE IF z≤x THEN x ELSE pred y ELSE IF IF pred y≤z THEN z ELSE IF z≤x %
THEN x ELSE pred y≤IF pred z≤x THEN x ELSE pred z THEN IF pred z≤x TH%
EN x ELSE pred z ELSE IF pred x≤y THEN y ELSE IF y≤z THEN z ELSE pred%
 x=IF y≤z THEN z ELSE x  (4)

*****TAUTEQ IF IF pred x≤y THEN y ELSE IF y≤z THEN z ELSE pred x≤IF p%
red y≤z THEN z ELSE IF z≤x THEN x ELSE pred y THEN IF pred y≤z THEN z%
 ELSE IF z≤x THEN x ELSE pred y ELSE IF IF pred y≤z THEN z ELSE IF z≤%
x THEN x ELSE pred y≤IF pred z≤x THEN x ELSE pred z THEN IF pred z≤x %
THEN x ELSE pred z ELSE IF pred x≤y THEN y ELSE IF y≤z THEN z ELSE pr%
ed x=IF y≤z THEN z ELSE x 14,41;

42 IF IF pred x≤y THEN y ELSE IF y≤z THEN z ELSE pred x≤IF pred y≤z T%
HEN z ELSE IF z≤x THEN x ELSE pred y THEN IF pred y≤z THEN z ELSE IF %
z≤x THEN x ELSE pred y ELSE IF IF pred y≤z THEN z ELSE IF z≤x THEN x %
ELSE pred y≤IF pred z≤x THEN x ELSE pred z THEN IF pred z≤x THEN x EL%
SE pred z ELSE IF pred x≤y THEN y ELSE IF y≤z THEN z ELSE pred x=IF y%
≤z THEN z ELSE x  (4)

*****REWRITE IF x≤y THEN y ELSE IF IF pred x≤y THEN y ELSE IF y≤z THE%
N z ELSE pred x≤IF pred y≤z THEN z ELSE IF z≤x THEN x ELSE pred y THE%
N IF pred y≤z THEN z ELSE IF z≤x THEN x ELSE pred y ELSE IF IF pred y%
≤z THEN z ELSE IF z≤x THEN x ELSE pred y≤IF pred z≤x THEN x ELSE IF x%
≤y THEN y ELSE pred z THEN IF pred z≤x THEN x ELSE IF x≤y THEN y ELSE%
 pred z ELSE IF pred x≤y THEN y ELSE IF y≤z THEN z ELSE pred x=IF x≤y%
 THEN y ELSE IF y≤z THEN z ELSE x  BY  5 LOGICTREE COMPTREE5;

43 IF x≤y THEN y ELSE IF IF pred x≤y THEN y ELSE IF y≤z THEN z ELSE p%
red x≤IF pred y≤z THEN z ELSE IF z≤x THEN x ELSE pred y THEN IF pred %
y≤z THEN z ELSE IF z≤x THEN x ELSE pred y ELSE IF IF pred y≤z THEN z %
ELSE IF z≤x THEN x ELSE pred y≤IF pred z≤x THEN x ELSE IF x≤y THEN y %
ELSE pred z THEN IF pred z≤x THEN x ELSE IF x≤y THEN y ELSE pred z EL%
SE IF pred x≤y THEN y ELSE IF y≤z THEN z ELSE pred x=IF x≤y THEN y EL%
SE IF y≤z THEN z ELSE x=IF IF pred x≤y THEN y ELSE IF y≤z THEN z ELSE%
 pred x≤IF pred y≤z THEN z ELSE IF z≤x THEN x ELSE pred y THEN IF pre%
```

```
d y≤z THEN z ELSE IF z≤x THEN x ELSE pred y ELSE IF IF pred y≤z THEN %
z ELSE IF z≤x THEN x ELSE pred y≤IF pred z≤x THEN x ELSE pred z THEN %
IF pred z≤x THEN x ELSE pred z ELSE IF pred x≤y THEN y ELSE IF y≤z TH%
EN z ELSE pred x=IF y≤z THEN z ELSE x  (4)

*****TAUT IF x≤y THEN y ELSE IF IF pred x≤y THEN y ELSE IF y≤z THEN z%
 ELSE pred x≤IF pred y≤z THEN z ELSE IF z≤x THEN x ELSE pred y THEN I%
F pred y≤z THEN z ELSE IF z≤x THEN x ELSE pred y ELSE IF IF pred y≤z %
THEN z ELSE IF z≤x THEN x ELSE pred y≤IF pred z≤x THEN x ELSE IF x≤y %
THEN y ELSE pred z THEN IF pred z≤x THEN x ELSE IF x≤y THEN y ELSE pr%
ed z ELSE IF pred x≤y THEN y ELSE IF y≤z THEN z ELSE pred x=IF x≤y TH%
EN y ELSE IF y≤z THEN z ELSE x 42,43;

44 IF x≤y THEN y ELSE IF IF pred x≤y THEN y ELSE IF y≤z THEN z ELSE p%
red x≤IF pred y≤z THEN z ELSE IF z≤x THEN x ELSE pred y THEN IF pred %
y≤z THEN z ELSE IF z≤x THEN x ELSE pred y ELSE IF IF pred y≤z THEN z %
ELSE IF z≤x THEN x ELSE pred y≤IF pred z≤x THEN x ELSE IF x≤y THEN y %
ELSE pred z THEN IF pred z≤x THEN x ELSE IF x≤y THEN y ELSE pred z EL%
SE IF pred x≤y THEN y ELSE IF y≤z THEN z ELSE pred x=IF x≤y THEN y EL%
SE IF y≤z THEN z ELSE x  (4)

*****⊃| 4⊃↑;

45 ¬(x≤y)⊃IF x≤y THEN y ELSE IF IF pred x≤y THEN y ELSE IF y≤z THEN z%
 ELSE pred x≤IF pred y≤z THEN z ELSE IF z≤x THEN x ELSE pred y THEN I%
F pred y≤z THEN z ELSE IF z≤x THEN x ELSE pred y ELSE IF IF pred y≤z %
THEN z ELSE IF z≤x THEN x ELSE pred y≤IF pred z≤x THEN x ELSE IF x≤y %
THEN y ELSE pred z THEN IF pred z≤x THEN x ELSE IF x≤y THEN y ELSE pr%
ed z ELSE IF pred x≤y THEN y ELSE IF y≤z THEN z ELSE pred x=IF x≤y TH%
EN y ELSE IF y≤z THEN z ELSE x

*****TAUTEQ IF x≤y THEN y ELSE IF IF pred x≤y THEN y ELSE IF y≤z THEN%
 z ELSE pred x≤IF pred y≤z THEN z ELSE IF z≤x THEN x ELSE pred y THEN%
 IF pred y≤z THEN z ELSE IF z≤x THEN x ELSE pred y ELSE IF IF pred y≤%
z THEN z ELSE IF z≤x THEN x ELSE pred y≤IF pred z≤x THEN x ELSE IF x≤%
y THEN y ELSE pred z THEN IF pred z≤x THEN x ELSE IF x≤y THEN y ELSE %
pred z ELSE IF pred x≤y THEN y ELSE IF y≤z THEN z ELSE pred x=IF x≤y %
THEN y ELSE IF y≤z THEN z ELSE x 3,45;

46 IF x≤y THEN y ELSE IF IF pred x≤y THEN y ELSE IF y≤z THEN z ELSE p%
red x≤IF pred y≤z THEN z ELSE IF z≤x THEN x ELSE pred y THEN IF pred %
y≤z THEN z ELSE IF z≤x THEN x ELSE pred y ELSE IF IF pred y≤z THEN z %
ELSE IF z≤x THEN x ELSE pred y≤IF pred z≤x THEN x ELSE IF x≤y THEN y %
ELSE pred z THEN IF pred z≤x THEN x ELSE IF x≤y THEN y ELSE pred z EL%
SE IF pred x≤y THEN y ELSE IF y≤z THEN z ELSE pred x=IF x≤y THEN y EL%
SE IF y≤z THEN z ELSE x

*****REWRITE tak1(x,y,z)=tak0(x,y,z)  BY  LOGICTREE COMPTREE BY { TAK%
1,TAK0};

47 tak1(x,y,z)=tak0(x,y,z)≡IF x≤y THEN y ELSE IF IF pred x≤y THEN y E%
LSE IF y≤z THEN z ELSE pred x≤IF pred y≤z THEN z ELSE IF z≤x THEN x E%
LSE pred y THEN IF pred y≤z THEN z ELSE IF z≤x THEN x ELSE pred y ELS%
E IF IF pred y≤z THEN z ELSE IF z≤x THEN x ELSE pred y≤IF pred z≤x TH%
EN x ELSE IF x≤y THEN y ELSE pred z THEN IF pred z≤x THEN x ELSE IF x%
```

```
≤y THEN y ELSE pred z ELSE IF pred x≤y THEN y ELSE IF y≤z THEN z ELSE%
 pred x=IF x≤y THEN y ELSE IF y≤z THEN z ELSE x

*****TAUT tak1(x,y,z)=tak0(x,y,z) 46,47;

48 tak1(x,y,z)=tak0(x,y,z)

*****VI ↑ x y z;

49 ∀x y z.tak1(x,y,z)=tak0(x,y,z)

*****
```

8. APPENDIX 2: RAMSEY'S THEOREM.

8.1. Introduction.

The following is Ramsey's theorem for denumerably infinite graphs.

RAMSEY'S THEOREM: If G is a complete, denumerable graph each of whose edges has been labeled *RED* or *BLACK*, then there is a complete, denumerable sub-graph of G whose edges are all of the same color.

PROOF: Let G*(1)*=G. For *i=1,2,.....*, repeat the following process: pick a point *x(i)*∈G(i); if *x(i)* is connected to infinitely many points of G*(i)* by red edges let G*(i+1)* be the set of points of G*(i)* that are connected to *x(i)* by red edges and label *x(i)* with *RED*, otherwise let G*(i+1)* be the set of points of G*(i)* that are connected to *x(i)* by black edges and label *x(i)* with *BLACK*. We see that, if G*(i)* is an infinite subset of G, so is G*(i+1)*, and all points of G*(i+1)* are connected to *x(i)* by edges of the color indicated by the label of *x(i)*; since G*(1)* is an infinite subset of G, so is G*(i)* for all *i*. Now consider the sequence *x(0),x(1),x(2),.....*; either infinitely many *x(i)* got labeled *RED* or infinitely many got *BLACK*. Those infinitely many *x(i)* that got the same label form an infinite one-colored sub-graph of G. QED.

Carrying out this proof in FOL or in GOAL is a non-trivial exercise. The first difficulty is at the logical level: choosing the correct way to express the iterative construction process using the axiom of choice, choosing some form of an axiom about the existence of inductively defined functions, and then bringing all these ends to match, requires painstaking attention to detail. In 1975 I constructed a FOL proof in 689 steps. The details and a commentary of the proof have been written up elsewhere [Weyhrauch et al. 1979]. This proof will be referred to as *the old proof* through this Appendix.

8.2. Axioms.

For the GOAL proof we are using the same axioms that were used in the earlier proof [Weyhrauch et al. 1979]. The rationale for this decision is that in this way the effectiveness of GOAL can be better appreciated.

8.2.1. General Axioms.

The following general axioms for Set Theory were written by Weyhrauch. They follow the spirit of Kelly's axiomatization in the Appendix to [Kelley 1955]. The *individual constant* λ stands for the *empty set*.

```
DECLARE PREDCONST ∈ 2[INF];
DECLARE PREDCONST ⊂ 2[INF];
DECLARE PREDCONST CONN 2[INF],ORD CARD NATNUM 1,WO(REL,*)[INF],CONG 2;
DECLARE PREDCONST REL FNC 1;
DECLARE PREDCONST SET 1;

DECLARE PREDPAR  A 1;

DECLARE OPCONST  ∪ 2[R←455 L←450];
DECLARE OPCONST  DOM,RNG(FNC)=*,MAPS 2,| 2[INF];
DECLARE OPCONST  MIN,SUP 1,CONV(REL)=REL,card(*)=ORD;
DECLARE OPCONST  EXP2 EXP3 1,CROSS 2,⊗ 2[INF];
DECLARE OPCONST  P 1;
DECLARE OPCONST  INTER 1[R←1000];
DECLARE OPCONST  \ 2[R←355,L←350];
DECLARE OPCONST  - 1[PRE];
DECLARE OPCONST  ∩ 2[R←555 L←550];
DECLARE OPCONST  UNION 1[R←1000];
DECLARE OPCONST  IMAGE(FNC,*);

DECLARE INDCONST E,ON,ALEPH0,omega;
DECLARE INDCONST V;
DECLARE INDCONST λ∈SET;

DECLARE INDVAR   a b c d e a1 b1 c1;
DECLARE INDVAR   u v w x y z∈SET;
DECLARE INDVAR   r s t∈REL f g h∈FNC;



DEFINE  SET:     ∀a.(SET(a)≡∃b.a∈b);;
AXIOM   KEXT:    ∀a b.(a=b≡∀c.(c∈a≡c∈b));;
AXIOM   KCOMP:   ∀a.(a∈{b|A(b)}≡SET(a)∧A(a));;

DEFINE  SUBSET:  ∀a b.(a⊂b≡∀c.(c∈a⊃c∈b));;
AXIOM   KPOWER:  ∀x.∃y.∀a.(a∈y≡a⊂x);;

DEFINE  union:   ∀a b.(a∪b={c|c∈a∨c∈b});;
AXIOM   kunion:  ∀x y.SET(x∪y);;

DEFINE  V:       V={a|a=a};;
DECLARE OPCONST singl 1;
DEFINE  UNIT:    ∀a.(singl(a)={c|a∈V∧c=a});;
DECLARE OPCONST pair 2;
```

```
DEFINE  PAIR:     ∀a b.(pair(a,b)=singl(a)usingl(b));;
DECLARE OPCONST opair 2;
DEFINE  OPAIR:    ∀a b.(opair(a,b)=pair(singl(a),pair(a,b)));;
DEFINE  TUPLE2:   ∀a b.(opair(a,b)=pair(singl(a),pair(a,b)));;
DECLARE OPCONST otriple 3;
DEFINE  TUPLE3:   ∀a b c.(otriple(a,b,c)=opair(a,opair(b,c)));;
DEFINE  REL:      ∀a.(REL(a)≡∀d.(d∈a⊃∃b c.(d=opair(b,c))));;
DEFINE  FNC:      ∀a.(FNC(a)≡REL(a)∧∀b c d.(opair(b,c)∈a∧opair(b,d)∈a⊃c=d));;
DEFINE  IMAGE:    ∀f a.(IMAGE(f,a)={c|∃y.(y∈a∧opair(y,c)∈f)});;
AXIOM   KSUBST:   ∀f y.SET(IMAGE(f,y));;

DEFINE  UNION:    ∀a.(UNION(a)={c|∃b.(b∈a∧c∈b)});;
AXIOM   KUNION:   ∀x.SET(UNION(x));;

DEFINE  EMPTY:    λ={a|¬a=a};;
DEFINE  inter:    ∀a b.(a∩b={c|c∈a∧c∈b});;
AXIOM   REG:      ∀a.(¬a=λ⊃∃y.(y∈a∧y∩a=λ));;
AXIOM   INF:      ∃x.(λ∈x∧∀y.(y∈x⊃yusingl(y)∈x));;


DEFINE  COMPL:    ∀a.(~a={c|¬c∈a});;
DEFINE  DIFF:     ∀a b.(a\b=a∩~b);;
DEFINE  INTER:    ∀a.(INTER(a)={c|∀b.(b∈a⊃c∈b)});;
DEFINE  POWER:    ∀a.(P(a)={c|c⊂a});;
DEFINE  EXP2:     ∀a.(EXP2(a)={c|∃x y.(x∈a∧y∈a∧c=opair(x,y))});;
DEFINE  EXP3:     ∀a.(EXP3(a)={c|∃x y z.(x∈a∧y∈a∧z∈a∧c=otriple(x,y,z))});;
DEFINE  CROSS:    ∀a b.(CROSS(a,b)={c|∃d e.(c=opair(d,e)∧d∈a∧e∈b)});;
DEFINE  COMPO:    ∀a b.(a⊛b=
                                  {c|∃a1 b1 c1.(c=opair(a1,b1)∧opair(a1,c1)∈a∧opair(c1,b1)∈b)});;
DEFINE  DOM:      ∀f.(DOM(f)={c|∃a.opair(c,a)∈f});;
DEFINE  RNG:      ∀f.(RNG(f)={c|∃a.opair(a,c)∈f});;
DEFINE  MAPS:     ∀a b.(MAPS(a,b)={f|FNC(f)∧DOM(f)=b∧RNG(f)=a});;
DEFINE  RESTR:    ∀f a.(f|a=f∩CROSS(a,V));;
DEFINE  E:        E={c|∃a b.(c=opair(a,b)∧a∈b)};;
DEFINE  CONN:     ∀r a.(r CONN a≡∀b c.(b∈a∧c∈a⊃opair(b,c)∈r∨opair(c,b)∈r∨b=c));;
DEFINE  ORD:      ∀a.(ORD(a)≡(E CONN a ∧ ∀b.(b∈a⊃b⊂a)));;
DEFINE  ON:       ON={c|ORD(c)};;
DEFINE  MIN:      ∀a.(MIN(a)=INTER(ON∪a)∩UNION(ON∩a));;
DEFINE  SUP:      ∀a.(SUP(a)=MIN({c|ON∩a⊂c}));;
DEFINE  CONV:     ∀r.(CONV(r)={c|∃a b.(c=opair(a,b)∧opair(b,a)∈r)});;
DEFINE  CONG:     ∀a b.(CONG(a,b)≡∃f.(FNC(f)∧FNC(CONV(f))∧DOM(f)=a∧RNG(f)=b));;
DEFINE  CARD:     ∀a.(CARD(a)≡(a⊂ON∧¬∃b.(b⊂a∧CONG(a,b))));;
DEFINE  ard:      ∀a.(card(a)=INTER({c|CARD(c)∧CONG(c,a)}));;
DEFINE  ^         ∀r a.(r WO a≡((r CONN a)∧∀b.(b⊂a∧¬b=λ⊃
                                  ∃x.(x∈b∧¬∃c.(c∈b∧¬c=x∧opair(c,x)∈r)∧∀d.(d∈a⊃opair(d,d)∈r)))));;
DEFINE  NATNU     ∀a.(NATNUM(a)≡ORD(a)∧(CONV(E) WO a));;
DEFINE  ALEPH0:   ℵ0={c|NATNUM(c)};;
DEFINE  omega:    om    ={c|NATNUM(c)};;
```

8.2.2. Special axioms.

The following more advanced principles of Set Theory were also postulated for our work on this theorem. Of these, APPLY, CHOICE, and INDUCTDEF were taken from **[Mendelson 1964]**. And the axiom EDGESET is simply a definition so that Ramsey's theorem can be stated in suggestive terms.

```
DECLARE PREDCONST  LT   NATNUM 2 [INF];        COMMENT: 'LESS THAN':
DECLARE PREDCONST DENUM 1;
DECLARE OPCONST SUC (NATNUM)=NATNUM;           COMMENT: SUCCESSOR:
DECLARE OPCONST " 2[INF];                  COMMENT: APPLY:
DECLARE OPCONST EDGESET 1;

DECLARE INDVAR G R B aa bb cc dd ee;
DECLARE INDVAR i j k<NATNUM;
DECLARE INDVAR p<FNC;


AXIOM INDUCTION: A(λ)∧∀i.(A(i)⊃A(SUC(i)))⊃∀i.A(i);;

AXIOM APPLY: ∀b a.((∃d.∀c.(d=c=opair(a,c)<b)⊃opair(a,b"a)<b)∧
                                        (¬∃d.∀c.(d=c=opair(a,c)<b)⊃b"a=λ));;

AXIOM INDUCTDEF: ∀x a.∃c.∀b.(c=b ≡ FNC(b)∧DOM(b)=omega∧b"λ=x
                                        ∧∀i.(b"SUC(i)=a"(b"i)));;

AXIOM CHOICE:∀x.∃f.∀a.(a⊂x∧¬a=λ⊃f"a<a);;

AXIOM EDGESET: ∀b.(EDGESET(b)={a|∃c d.(c<b∧d<b∧¬c=d∧a=pair(c,d))});;

AXIOM DENUM:∀a.(DENUM(a)≡CONG(omega,a));;

AXIOM SUC: ∀i.¬λ=SUC(i),∀i j.(SUC(i)=SUC(j)⊃i=j);;
```

8.2.3. Auxiliary lemmas.

The following auxiliary lemmas are a subset of those that were postulated for the earlier proof **[Weyhrauch et al. 1979]**. The first three concern the relation *less than* (LT).

```
AXIOM LESS2: ∀i j.(¬i=j≡i LT j ∨ j LT i);;
AXIOM LESS4: ∀i.¬i LT λ;;
AXIOM LESS7: ∀i j.(i LT SUC(j)≡i=j∨i LT j);;


MG SET≥{NATNUM};
MG REL≥{FNC};
AXIOM AUX1: ∀a.(DENUM(a)⊃SET(a));;
AXIOM AUX2: ∀a.(a⊂a);;
AXIOM AUX3: ∀a.(DENUM(a)⊃¬a=λ);;
AXIOM AUX4: ∀a b.(a∈b⊃SET(a));;
AXIOM AUX5: ∀a b.(SET(opair(a,b))≡SET(a)∧SET(b));;
AXIOM AUX6: ∀a b c.(a∈(b∪c)≡a∈b∨a∈c);;
AXIOM AUX9: ∀a b.(DENUM(a)⊃DENUM(a\singl(b)));;
AXIOM AUX10: ∀a b.(DENUM(a∪b)⊃DENUM(a)∨DENUM(b));;
AXIOM AUX11: ∀x b c.(c∈(b\singl(x))≡c∈b∧¬c=x);;
AXIOM AUX12: ∀a b c.(pair(a,b)∈EDGESET(c)≡a∈c∧b∈c∧¬a=b);;
AXIOM AUX13: ∀a b c d.(opair(a,b)=opair(c,d)≡a=c∧b=d);;
AXIOM AUX18: ∀a b.(a⊂b⊃a∩b=a);;
AXIOM AUX20:∀a b.(a∩b⊂a∧a∩b⊂b);;
AXIOM AUX22: ∀a b.(a∩b=b∩a);;
AXIOM AUX23: ∀a b c.(a⊂b∧b⊂c⊃a⊂c);;
AXIOM AUX24: ∀a b.(pair(a,b)=pair(b,a));;
AXIOM AUX25: ∀a b c.(a∈b∩c≡a∈b∧a∈c);;
AXIOM AUX27:∀a b c d.(opair(a,b)=opair(c,d)≡a=c∧b=d);;
AXIOM AUX28: ∀a b c.(a∈b∧b⊂c⊃a∈c);;
AXIOM AUX29: ∀a b.(a\b⊂a);;
AXIOM AUX30: DENUM(omega);;
AXIOM AUX34:∀a b.(DENUM(a)∧CONG(a,b)⊃DENUM(b));;
AXIOM AUX35: ∀a b.(a⊂a∪b∧b⊂a∪b);;
```

8.3. Proofs of some auxiliary theorems.

The first 184 lines of the earlier proof [Weyhrauch et al. 1979] proved several set theoretic facts. For the GOAL proof we have used a subset of these. In this section we shall show an independent proof of those. Later they will be postulated for the main proof.

The total number of commands used in the following proofs is 39: this figure includes both the *forward* proof steps using FOL commands and the calls to TRY. If we add the commands that create the goals, that is five instances of the GOAL command, then we come to a total of 44. In the old proof, this same set of facts required 184 lines. Thus we achieve a fourfold reduction in the number of commands, for this particular set of lemmas.

8.3.1. Restriction of a function.

The first lemma says that the *restriction* of a *function* is again a *function*. The old proof required 27 lines. The following GOAL proof uses only eight instances of the TRY command, and generates a FOL proof of 27 lines.

```
*****GOAL ∀f a.FNC(f | a);

Goal *1: ∀f a.FNC(f | a)

*****TRY USING REWRITE BY {RESTR};

Goal *1*1: ∀f a.FNC(f∩CROSS(a,V))

*****TRY USING ∀I a1 a;

Goal *1*1*1: FNC(a1)⊃FNC(a1∩CROSS(a,V))

*****TRY USING REWRITE BY {FNC REL AUX25};

Goal *1*1*1*1: (∀d.(d∈a1⊃∃b c.d=opair(b,c))∧∀b c d.((opair(b,c)∈a1∧o%
pair(b,d)∈a1)⊃c=d))⊃(∀d.((d∈a1∧d∈CROSS(a,V))⊃∃b c.d=opair(b,c))∧∀b c %
d.(((opair(b,c)∈a1∧opair(b,c)∈CROSS(a,V))∧(opair(b,d)∈a1∧opair(b,d)∈C%
ROSS(a,V)))⊃c=d))

*****TRY USING ELIMINATION DEPTH 4;

Goal *1*1*1*1*1: ∀d.((d∈a1∧d∈CROSS(a,V))⊃∃b c.d=opair(b,c))∧∀b c d.(%
((opair(b,c)∈a1∧opair(b,c)∈CROSS(a,V))∧(opair(b,d)∈a1∧opair(b,d)∈CROS%
S(a,V)))⊃c=d)

1 ∀d.(d∈a1⊃∃b c.d=opair(b,c))∧∀b c d.((opair(b,c)∈a1∧opair(b,d)∈a1)⊃c%
=d)  (1)

2 ∀b c d.((opair(b,c)∈a1∧opair(b,d)∈a1)⊃c=d)  (1)

3 ∀d.(d∈a1⊃∃b c.d=opair(b,c))  (1)

Goal *1*1*1*1*1*1: ∀d.((d∈a1∧d∈CROSS(a,V))⊃∃b c.d=opair(b,c))
Goal *1*1*1*1*1*2: ∀b c d.(((opair(b,c)∈a1∧opair(b,c)∈CROSS(a,V))∧(o%
pair(b,d)∈a1∧opair(b,d)∈CROSS(a,V)))⊃c=d)
Goal *1*1*1*1*1*1*1: (d∈a1∧d∈CROSS(a,V))⊃∃b c.d=opair(b,c)
Goal *1*1*1*1*1*1*1*1: ∃b c.d=opair(b,c)
Goal *1*1*1*1*1*2*1: ((opair(b,c)∈a1∧opair(b,c)∈CROSS(a,V))∧(opair(b%
,d)∈a1∧opair(b,d)∈CROSS(a,V)))⊃c=d
Goal *1*1*1*1*1*2*1*1: c=d

*****TRY USING IMPLICATION;

4 (opair(b,c)∈a1∧opair(b,c)∈CROSS(a,V))∧(opair(b,d)∈a1∧opair(b,d)∈CRO%
```

SS(a,V)) (4)

5 opair(b,d)∈CROSS(a,V) (4)

6 opair(b,d)∈a1 (4)

7 opair(b,c)∈CROSS(a,V) (4)

8 opair(b,c)∈a1 (4)

Goal #1#1#1#1#1#2#1#1#1: opair(b,c)∈a1∧opair(b,d)∈a1

*****TRY USING TAUT;

9 opair(b,c)∈a1∧opair(b,d)∈a1 (1 4)

RESOLVE (opair(b,c)∈a1∧opair(b,d)∈a1)⊃c=d , opair(b,c)∈a1∧opair(b,d)∈%
a1 ↠ c=d

10 c=d (1 4)

11 ((opair(b,c)∈a1∧opair(b,c)∈CROSS(a,V))∧(opair(b,d)∈a1∧opair(b,d)∈C%
ROSS(a,V)))⊃c=d (1)

12 ∀b c d.(((opair(b,c)∈a1∧opair(b,c)∈CROSS(a,V))∧(opair(b,d)∈a1∧opai%
r(b,d)∈CROSS(a,V)))⊃c=d) (1)

*****TRY USING IMPLICATION;

13 d∈a1∧d∈CROSS(a,V) (13)

14 d∈CROSS(a,V) (13)

15 d∈a1 (13)

Goal #1#1#1#1#1#1#1#1#1: d∈a1

*****TRY USING TAUT;

16 d∈a1 (1 13)

RESOLVE d∈a1⊃∃b c.d=opair(b,c) , d∈a1 ↠ ∃b c.d=opair(b,c)

17 ∃b c.d=opair(b,c) (1 13)

18 (d∈a1∧d∈CROSS(a,V))⊃∃b c.d=opair(b,c) (1)

19 ∀d.((d∈a1∧d∈CROSS(a,V))⊃∃b c.d=opair(b,c)) (1)

20 ∀d.((d∈a1∧d∈CROSS(a,V))⊃∃b c.d=opair(b,c))∧∀b c d.(((opair(b,c)∈a1%
∧opair(b,c)∈CROSS(a,V))∧(opair(b,d)∈a1∧opair(b,d)∈CROSS(a,V)))⊃c=d) %
(1)

```
21 (∀d.(d∈a1⊃∃b c.d=opair(b,c))∧∀b c d.((opair(b,c)∈a1∧opair(b,d)∈a1)%
⊃c=d))⊃(∀d.((d∈a1∧d∈CROSS(a,V))⊃∃b c.d=opair(b,c))∧∀b c d.(((opair(b,%
c)∈a1∧opair(b,c)∈CROSS(a,V))∧(opair(b,d)∈a1∧opair(b,d)∈CROSS(a,V)))⊃c%
=d))

22 (FNC(a1)⊃FNC(a1∩CROSS(a,V)))≡((∀d.(d∈a1⊃∃b c.d=opair(b,c))∧∀b c d.%
((opair(b,c)∈a1∧opair(b,d)∈a1)⊃c=d))⊃(∀d.((d∈a1∧d∈CROSS(a,V))⊃∃b c.d=%
opair(b,c))∧∀b c d.(((opair(b,c)∈a1∧opair(b,c)∈CROSS(a,V))∧(opair(b,d%
)∈a1∧opair(b,d)∈CROSS(a,V)))⊃c=d)))

23 FNC(a1)⊃FNC(a1∩CROSS(a,V))

24 ∀f a.(FNC(f)⊃FNC(f∩CROSS(a,V)))

25 ∀f a.FNC(f∩CROSS(a,V))

26 ∀f a.FNC(f | a)≡∀f a.FNC(f∩CROSS(a,V))

27 ∀f a.FNC(f | a)

*****
```

8.3.2. Domain of the restriction.

The next lemma says that if we *restrict* a *function* to a subset of its *domain*, the *domain* of the *restriction* is equal to that subset. The GOAL proof takes nine instances of TRY, one call to the QED command, plus four forward proving commands: two universal specializations, one call to RESOLVE and one to REWRITE. The old proof was in 54 lines.

```
*****GOAL ∀f a.(a⊂DOM(f)⊃DOM(f | a)=a);

Goal #2: ∀f a.(a⊂DOM(f)⊃DOM(f | a)=a)

*****TRY USING REWRITE BY {SUBSET KEXT};

Goal #2#1: ∀f a.(∀c.(c∈a⊃c∈DOM(f))⊃∀c.(c∈DOM(f | a)≡c∈a))

*****VE DOM f | a;

28 FNC(f | a)⊃DOM(f | a)={c|∃a1.opair(c,a1)∈(f | a)}

*****RESOLVE † L41;

RESOLVE FNC(f | a)⊃DOM(f | a)={c|∃a1.opair(c,a1)∈(f | a)} , ∀f a.FNC(%
f | a) →→ DOM(f | a)={c|∃a1.opair(c,a1)∈(f | a)}
```

```
29 DOM(f | a)={c|∃a1.opair(c,a1)∈(f | a)}

*****TRY USING REWRITE BY { ↑ AUX5 AUX13 AUX25 V CROSS SET DOM RESTR};

Goal *2*1*1: ∀f a2.(∀c.(c∈a2⊃(∃b.c∈b∧∃a.opair(c,a)∈f))⊃∀c.((∃b.c∈b∧∃%
a1.(opair(c,a1)∈f∧((∃b.c∈b∧∃b.a1∈b)∧∃d e.((c=d∧a1=e)∧(d∈a2∧∃b.e∈b))))%
)≡c∈a2))

*****TRY USING ELIMINATION DEPTH 8;

Goal *2*1*1*1: ∀c.(c∈a2⊃(∃b.c∈b∧∃a.opair(c,a)∈f))⊃∀c.((∃b.c∈b∧∃a1.(o%
pair(c,a1)∈f∧((∃b.c∈b∧∃b.a1∈b)∧∃d e.((c=d∧a1=e)∧(d∈a2∧∃b.e∈b)))))≡c∈a%
2)
Goal *2*1*1*1*1: ∀c.((∃b.c∈b∧∃a1.(opair(c,a1)∈f∧((∃b.c∈b∧∃b.a1∈b)∧∃d%
 e.((c=d∧a1=e)∧(d∈a2∧∃b.e∈b)))))≡c∈a2)
30 ∀c.(c∈a2⊃(∃b.c∈b∧∃a.opair(c,a)∈f))  (30)

Goal *2*1*1*1*1*1: (∃b.c∈b∧∃a1.(opair(c,a1)∈f∧((∃b.c∈b∧∃b.a1∈b)∧∃d e%
.((c=d∧a1=e)∧(d∈a2∧∃b.e∈b)))))≡c∈a2
Goal *2*1*1*1*1*1*1: (∃b.c∈b∧∃a1.(opair(c,a1)∈f∧((∃b.c∈b∧∃b.a1∈b)∧∃d%
 e.((c=d∧a1=e)∧(d∈a2∧∃b.e∈b)))))⊃c∈a2
Goal *2*1*1*1*1*1*2: c∈a2⊃(∃b.c∈b∧∃a1.(opair(c,a1)∈f∧((∃b.c∈b∧∃b.a1∈%
b)∧∃d e.((c=d∧a1=e)∧(d∈a2∧∃b.e∈b)))))
Goal *2*1*1*1*1*1*1*1: c∈a2
Goal *2*1*1*1*1*1*2*1: ∃b.c∈b∧∃a1.(opair(c,a1)∈f∧((∃b.c∈b∧∃b.a1∈b)∧∃%
d e.((c=d∧a1=e)∧(d∈a2∧∃b.e∈b))))
31 c∈a2  (31)

RESOLVE c∈a2⊃(∃b.c∈b∧∃a.opair(c,a)∈f) , c∈a2 ↠ ∃b.c∈b∧∃a.opair(c,a)∈%
f

32 ∃b.c∈b∧∃a.opair(c,a)∈f  (30 31)

33 ∃a.opair(c,a)∈f  (30 31)

34 ∃b.c∈b  (30 31)

Goal *2*1*1*1*1*1*2*1*1: ∃b.c∈b
Goal *2*1*1*1*1*1*2*1*2: ∃a1.(opair(c,a1)∈f∧((∃b.c∈b∧∃b.a1∈b)∧∃d e.(%
(c=d∧a1=e)∧(d∈a2∧∃b.e∈b))))
Goal *2*1*1*1*1*1*2*1*1*1: c∈b
Goal *2*1*1*1*1*1*2*1*2*1: opair(c,a1)∈f∧((∃b.c∈b∧∃b.a1∈b)∧∃d e.((c=%
d∧a1=e)∧(d∈a2∧∃b.e∈b)))
Goal *2*1*1*1*1*1*2*1*2*1*1: opair(c,a1)∈f
Goal *2*1*1*1*1*1*2*1*2*1*2: (∃b.c∈b∧∃b.a1∈b)∧∃d e.((c=d∧a1=e)∧(d∈a2%
∧∃b.e∈b))

*****QED *2*1*1*1*1*1*2*1*1 34;

*****TRY *2*1*1*1*1*1*2*1*2*1*1 USING UNIFY 33;

35 ∃a1.opair(c,a1)∈f  (30 31)

36 opair(c,a1)∈f  (36)
```

```
*****REWRITE AUX5 BY {SET};

3 substitutions were made

37 ∀a b1.(∃b.opair(a,b1)∈b≡(∃b.a∈b∧∃b.b1∈b))

*****∀E ↑ c a1;

38 ∃b.opair(c,a1)∈b≡(∃b.c∈b∧∃b.a1∈b)

*****TRY *2*1*1*1*1*1*2*1*2*1*2 USING ∧I;

Goal *2*1*1*1*1*1*2*1*2*1*2*1: ∃b.c∈b∧∃b.a1∈b
Goal *2*1*1*1*1*1*2*1*2*1*2*2: ∃d e.((c=d∧a1=e)∧(d∈a2∧∃b.e∈b))

*****TRY 1 USING MONADIC 36 38;

39 ∃b.c∈b∧∃b.a1∈b  (36)

*****TRY USING MONADIC 31 ↑;

40 ∃b.a1∈b  (36)

41 ∃b.c∈b  (30 31)

42 ∃d e.((c=d∧a1=e)∧(d∈a2∧∃b.e∈b))  (31 36)

43 (∃b.c∈b∧∃b.a1∈b)∧∃d e.((c=d∧a1=e)∧(d∈a2∧∃b.e∈b))  (31 36)

44 opair(c,a1)∈f∧((∃b.c∈b∧∃b.a1∈b)∧∃d e.((c=d∧a1=e)∧(d∈a2∧∃b.e∈b)))  %
(31 36)

45 ∃a1.(opair(c,a1)∈f∧((∃b.c∈b∧∃b.a1∈b)∧∃d e.((c=d∧a1=e)∧(d∈a2∧∃b.e∈b%
))))  (30 31)

46 ∃b.c∈b∧∃a1.(opair(c,a1)∈f∧((∃b.c∈b∧∃b.a1∈b)∧∃d e.((c=d∧a1=e)∧(d∈a2%
∧∃b.e∈b))))  (30 31)

47 c∈a2⊃(∃b.c∈b∧∃a1.(opair(c,a1)∈f∧((∃b.c∈b∧∃b.a1∈b)∧∃d e.((c=d∧a1=e)%
∧(d∈a2∧∃b.e∈b)))))  (30)

*****TRY USING LOGIC;

48 ∃b.c∈b∧∃a1.(opair(c,a1)∈f∧((∃b.c∈b∧∃b.a1∈b)∧∃d e.((c=d∧a1=e)∧(d∈a2%
∧∃b.e∈b))))  (48)

49 ∃a1.(opair(c,a1)∈f∧((∃b.c∈b∧∃b.a1∈b)∧∃d e.((c=d∧a1=e)∧(d∈a2∧∃b.e∈b%
))))  (48)

50 ∃b.c∈b  (48)

51 ∃a1 d e.((c=d∧a1=e)∧(d∈a2∧∃b.e∈b))  (48)
```

```
52 ∃a1 b.a1∈b  (48)

53 ∃a1 b.c∈b  (48)

54 ∃a1.opair(c,a1)∈f  (48)

55 ∃a1 d e b.e∈b  (48)

56 ∃a1 d e.d∈a2  (48)

57 ∃a1 d e.a1=e  (48)

58 ∃a1 d e.c=d  (48)

We have a failqueue of length:  1
Starting a new 2-th pass on new queue of length: 1
We have a failqueue of length:  1
Failure: can't prove anything on failqueue.
The tactic LOGIC can't be applied to goal
Goal *2*1*1*1*1*1*1*1: c∈a2
    IMPVL:  48  ∃b.c∈b∧∃a1.(opair(c,a1)∈f∧((∃b.c∈b∧∃b.a1∈b)∧∃d e.((%
c=d∧a1=e)∧(d∈a2∧∃b.e∈b))))
    FACTS:  30  ∀c.(c∈a2⊃(∃b.c∈b∧∃a.opair(c,a)∈f))
        55  ∃a1 d e b.e∈b
        56  ∃a1 d e.d∈a2
        57  ∃a1 d e.a1=e
        58  ∃a1 d e.c=d
        51  ∃a1 d e.((c=d∧a1=e)∧(d∈a2∧∃b.e∈b))
        52  ∃a1 b.a1∈b
        53  ∃a1 b.c∈b
        54  ∃a1.opair(c,a1)∈f
        49  ∃a1.(opair(c,a1)∈f∧((∃b.c∈b∧∃b.a1∈b)∧∃d e.((c=d∧a1=e)∧%
(d∈a2∧∃b.e∈b))))
        50  ∃b.c∈b
        47  c∈a2⊃(∃b.c∈b∧∃a1.(opair(c,a1)∈f∧((∃b.c∈b∧∃b.a1∈b)∧∃d e%
.((c=d∧a1=e)∧(d∈a2∧∃b.e∈b)))))
        48  ∃b.c∈b∧∃a1.(opair(c,a1)∈f∧((∃b.c∈b∧∃b.a1∈b)∧∃d e.((c=d%
∧a1=e)∧(d∈a2∧∃b.e∈b))))
      Simpsets:  ( BY  LOGICTREE  COMPTREE)
      Quantelimlist:  ((c ∀) (a2 ∀) (f ∀))

*****TRY USING MONADIC 51;

59 c∈a2  (48)

60 (∃b.c∈b∧∃a1.(opair(c,a1)∈f∧((∃b.c∈b∧∃b.a1∈b)∧∃d e.((c=d∧a1=e)∧(d∈a%
2∧∃b.e∈b)))))⊃c∈a2

61 (∃b.c∈b∧∃a1.(opair(c,a1)∈f∧((∃b.c∈b∧∃b.a1∈b)∧∃d e.((c=d∧a1=e)∧(d∈a%
2∧∃b.e∈b)))))≡c∈a2  (30)

62 ∀c.((∃b.c∈b∧∃a1.(opair(c,a1)∈f∧((∃b.c∈b∧∃b.a1∈b)∧∃d e.((c=d∧a1=e)∧%
(d∈a2∧∃b.e∈b)))))≡c∈a2)  (30)
```

```
63 ∀c.(c∈a2⊃(∃b.c∈b∧∃a.opair(c,a)∈f))⊃∀c.((∃b.c∈b∧∃a1.(opair(c,a1)∈f∧%
((∃b.c∈b∧∃b.a1∈b)∧∃d e.((c=d∧a1=e)∧(d∈a2∧∃b.e∈b)))))≡c∈a2)

64 ∀f a2.(∀c.(c∈a2⊃(∃b.c∈b∧∃a.opair(c,a)∈f))⊃∀c.((∃b.c∈b∧∃a1.(opair(c%
,a1)∈f∧((∃b.c∈b∧∃b.a1∈b)∧∃d e.((c=d∧a1=e)∧(d∈a2∧∃b.e∈b)))))≡c∈a2))

65 ∀f a.(∀c.(c∈a⊃c∈DOM(f))⊃∀c.(c∈DOM(f | a)≡c∈a))≡∀f a2.(∀c.(c∈a2⊃(∃b%
.c∈b∧∃a.opair(c,a)∈f))⊃∀c.((∃b.c∈b∧∃a1.(opair(c,a1)∈f∧((∃b.c∈b∧∃b.a1∈%
b)∧∃d e.((c=d∧a1=e)∧(d∈a2∧∃b.e∈b)))))≡c∈a2))

66 ∀f a.(∀c.(c∈a⊃c∈DOM(f))⊃∀c.(c∈DOM(f | a)≡c∈a))

67 ∀f a.(a⊂DOM(f)⊃DOM(f | a)=a)≡∀f a.(∀c.(c∈a⊃c∈DOM(f))⊃∀c.(c∈DOM(f |%
 a)≡c∈a))

68 ∀f a.(a⊂DOM(f)⊃DOM(f | a)=a)

*****
```

8.3.3. Restriction of a one-to-one function.

The next lemma states (in somewhat different terms) that the *restriction* of an *one-to-one* function is again *one-to-one*. The old proof took 58 steps. The following one requires six calls to TRY, one to RETRY, two calls to QED, and the following four *forward* commands from FOL: two universal specializations, one REWRITE, and one call to TAUT. A total of 13 commands instead 58.

```
*****GOAL ∀f a.(FNC(CONV(f))⊃FNC(CONV(f | a)));

Goal #3:  ∀f a.(FNC(CONV(f))⊃FNC(CONV(f | a)))

*****VE CONV f | a;

54 REL(f | a)⊃CONV(f | a)={c|∃a1 b.(c=opair(a1,b)∧opair(b,a1)∈(f | a)%
)}

*****REWRITE L41 BY {FNC};

1 substitutions were made

55 ∀f a.(REL(f | a)∧∀b c d.((opair(b,c)∈(f | a)∧opair(b,d)∈(f | a))⊃c%
=d))

*****VE ↑ f a;

56 REL(f | a)∧∀b c d.((opair(b,c)∈(f | a)∧opair(b,d)∈(f | a))⊃c=d)
```

```
*****TAUT :#2 ,↑;

57 CONV(f | a)={c|∃a1 b.(c=opair(a1,b)∧opair(b,a1)∈(f | a))}

*****TRY USING REWRITE BY { ↑ FNC REL CONV RESTR AUX13 AUX25};

Goal #3#1: ∀f a2.((∀d.((SET(d)∧∃a b.(d=opair(a,b)∧opair(b,a)∈f))⊃∃b %
c.d=opair(b,c))∧∀b1 c d.(((SET(opair(b1,c))∧∃a b.((b1=a∧c=b)∧opair(b,%
a)∈f))∧(SET(opair(b1,d))∧∃a b.((b1=a∧d=b)∧opair(b,a)∈f)))⊃c=d))⊃(∀d.(%
(SET(d)∧∃a1 b.(d=opair(a1,b)∧(opair(b,a1)∈f∧opair(b,a1)∈CROSS(a2,V)))%
)⊃∃b c.d=opair(b,c))∧∀b1 c d.(((SET(opair(b1,c))∧∃a1 b.((b1=a1∧c=b)∧(%
opair(b,a1)∈f∧opair(b,a1)∈CROSS(a2,V))))∧(SET(opair(b1,d))∧∃a1 b.((b1%
=a1∧d=b)∧(opair(b,a1)∈f∧opair(b,a1)∈CROSS(a2,V)))))⊃c=d)))

*****TRY USING ELIMINATION DEPTH 5;

Goal #3#1#1: (∀d.((SET(d)∧∃a b.(d=opair(a,b)∧opair(b,a)∈f))⊃∃b c.d=o%
pair(b,c))∧∀b1 c d.(((SET(opair(b1,c))∧∃a b.((b1=a∧c=b)∧opair(b,a)∈f)%
)∧(SET(opair(b1,d))∧∃a b.((b1=a∧d=b)∧opair(b,a)∈f)))⊃c=d))⊃(∀d.((SET(%
d)∧∃a1 b.(d=opair(a1,b)∧(opair(b,a1)∈f∧opair(b,a1)∈CROSS(a2,V))))⊃∃b %
c.d=opair(b,c))∧∀b1 c d.(((SET(opair(b1,c))∧∃a1 b.((b1=a1∧c=b)∧(opair%
(b,a1)∈f∧opair(b,a1)∈CROSS(a2,V))))∧(SET(opair(b1,d))∧∃a1 b.((b1=a1∧d%
=b)∧(opair(b,a1)∈f∧opair(b,a1)∈CROSS(a2,V)))))⊃c=d))
Goal #3#1#1#1: ∀d.((SET(d)∧∃a1 b.(d=opair(a1,b)∧(opair(b,a1)∈f∧opair%
(b,a1)∈CROSS(a2,V))))⊃∃b c.d=opair(b,c))∧∀b1 c d.(((SET(opair(b1,c))∧%
∃a1 b.((b1=a1∧c=b)∧(opair(b,a1)∈f∧opair(b,a1)∈CROSS(a2,V))))∧(SET(opa%
ir(b1,d))∧∃a1 b.((b1=a1∧d=b)∧(opair(b,a1)∈f∧opair(b,a1)∈CROSS(a2,V)))%
))⊃c=d)

58 ∀d.((SET(d)∧∃a b.(d=opair(a,b)∧opair(b,a)∈f))⊃∃b c.d=opair(b,c))∧∀%
b1 c d.(((SET(opair(b1,c))∧∃a b.((b1=a∧c=b)∧opair(b,a)∈f))∧(SET(opair%
(b1,d))∧∃a b.((b1=a∧d=b)∧opair(b,a)∈f)))⊃c=d)  (58)

59 ∀b1 c d.(((SET(opair(b1,c))∧∃a b.((b1=a∧c=b)∧opair(b,a)∈f))∧(SET(o%
pair(b1,d))∧∃a b.((b1=a∧d=b)∧opair(b,a)∈f)))⊃c=d)  (58)

60 ∀d.((SET(d)∧∃a b.(d=opair(a,b)∧opair(b,a)∈f))⊃∃b c.d=opair(b,c))  %
(58)

Goal #3#1#1#1#1: ∀d.((SET(d)∧∃a1 b.(d=opair(a1,b)∧(opair(b,a1)∈f∧opa%
ir(b,a1)∈CROSS(a2,V))))⊃∃b c.d=opair(b,c))
Goal #3#1#1#1#2: ∀b1 c d.(((SET(opair(b1,c))∧∃a1 b.((b1=a1∧c=b)∧(opa%
ir(b,a1)∈f∧opair(b,a1)∈CROSS(a2,V))))∧(SET(opair(b1,d))∧∃a1 b.((b1=a1%
∧d=b)∧(opair(b,a1)∈f∧opair(b,a1)∈CROSS(a2,V)))))⊃c=d)
Goal #3#1#1#1#1#1: (SET(d)∧∃a1 b.(d=opair(a1,b)∧(opair(b,a1)∈f∧opair%
(b,a1)∈CROSS(a2,V))))⊃∃b c.d=opair(b,c)
Goal #3#1#1#1#1#1#1: ∃b c.d=opair(b,c)
Goal #3#1#1#1#2#1: ((SET(opair(b1,c))∧∃a1 b.((b1=a1∧c=b)∧(opair(b,a1%
)∈f∧opair(b,a1)∈CROSS(a2,V))))∧(SET(opair(b1,d))∧∃a1 b.((b1=a1∧d=b)∧(%
opair(b,a1)∈f∧opair(b,a1)∈CROSS(a2,V)))))⊃c=d
Goal #3#1#1#1#2#1#1: c=d

*****RETRY #3#1#1#1#1 USING MONADIC;

Goal #3#1#1#1#1: ∀d.((SET(d)∧∃a1 b.(d=opair(a1,b)∧(opair(b,a1)∈f∧opa%
```

ir(b,a1)∈CROSS(a2,V))))⊃∃b c.d=opair(b,c)) abandoned.

61 ∀d.((SET(d)∧∃a1 b.(d=opair(a1,b)∧(opair(b,a1)∈f∧opair(b,a1)∈CROSS(%
a2,V))))⊃∃b c.d=opair(b,c))

*****TRY USING IMPLICATION;

62 (SET(opair(b1,c))∧∃a1 b.((b1=a1∧c=b)∧(opair(b,a1)∈f∧opair(b,a1)∈CR%
OSS(a2,V))))∧(SET(opair(b1,d))∧∃a1 b.((b1=a1∧d=b)∧(opair(b,a1)∈f∧opai%
r(b,a1)∈CROSS(a2,V)))) (62)

63 ∃a1 b.((b1=a1∧d=b)∧(opair(b,a1)∈f∧opair(b,a1)∈CROSS(a2,V))) (62)

64 SET(opair(b1,d)) (62)

65 ∃a1 b.((b1=a1∧c=b)∧(opair(b,a1)∈f∧opair(b,a1)∈CROSS(a2,V))) (62)

66 SET(opair(b1,c)) (62)

67 ∃a1 b.opair(b,a1)∈CROSS(a2,V) (62)

68 ∃a1 b.opair(b,a1)∈f (62)

69 ∃a1 b.c=b (62)

70 ∃a1 b.b1=a1 (62)

71 ∃a1 b.opair(b,a1)∈CROSS(a2,V) (62)

72 ∃a1 b.opair(b,a1)∈f (62)

73 ∃a1 b.d=b (62)

74 ∃a1 b.b1=a1 (62)

Goal *3*1*1*1*2*1*1*1: (SET(opair(b1,c))∧∃a b.((b1=a∧c=b)∧opair(b,a)%
∈f))∧(SET(opair(b1,d))∧∃a b.((b1=a∧d=b)∧opair(b,a)∈f))

*****TRY USING ELIMINATION DEPTH 2;

Goal *3*1*1*1*2*1*1*1*1: SET(opair(b1,c))∧∃a b.((b1=a∧c=b)∧opair(b,a%
)∈f)
Goal *3*1*1*1*2*1*1*1*2: SET(opair(b1,d))∧∃a b.((b1=a∧d=b)∧opair(b,a%
)∈f)
Goal *3*1*1*1*2*1*1*1*1*1: SET(opair(b1,c))
Goal *3*1*1*1*2*1*1*1*1*2: ∃a b.((b1=a∧c=b)∧opair(b,a)∈f)
Goal *3*1*1*1*2*1*1*1*2*1: SET(opair(b1,d))
Goal *3*1*1*1*2*1*1*1*2*2: ∃a b.((b1=a∧d=b)∧opair(b,a)∈f)

*****TRY USING MONADIC 63;

75 ∃a b.((b1=a∧d=b)∧opair(b,a)∈f) (62)

*****TRY *3*1*1*1*2*1*1*1*1*2 USING MONADIC 65;

76 ∃a b.((b1=a∧c=b)∧opair(b,a)∈f) (62)

*****QED *3*1*1*1*2*1*1*1*1*1 66;

77 SET(opair(b1,c))∧∃a b.((b1=a∧c=b)∧opair(b,a)∈f) (62)

*****QED 64;

78 SET(opair(b1,d))∧∃a b.((b1=a∧d=b)∧opair(b,a)∈f) (62)

79 (SET(opair(b1,c))∧∃a b.((b1=a∧c=b)∧opair(b,a)∈f))∧(SET(opair(b1,d)%
)∧∃a b.((b1=a∧d=b)∧opair(b,a)∈f)) (62)

RESOLVE ((SET(opair(b1,c))∧∃a b.((b1=a∧c=b)∧opair(b,a)∈f))∧(SET(opair%
(b1,d))∧∃a b.((b1=a∧d=b)∧opair(b,a)∈f))⊃c=d , (SET(opair(b1,c))∧∃a b%
.((b1=a∧c=b)∧opair(b,a)∈f))∧(SET(opair(b1,d))∧∃a b.((b1=a∧d=b)∧opair(%
b,a)∈f)) →→ c=d

80 c=d (58 62)

81 ((SET(opair(b1,c))∧∃a1 b.((b1=a1∧c=b)∧(opair(b,a1)∈f∧opair(b,a1)∈C%
ROSS(a2,V))))∧(SET(opair(b1,d))∧∃a1 b.((b1=a1∧d=b)∧(opair(b,a1)∈f∧opa%
ir(b,a1)∈CROSS(a2,V)))))⊃c=d (58)

82 ∀b1 c d.(((SET(opair(b1,c))∧∃a1 b.((b1=a1∧c=b)∧(opair(b,a1)∈f∧opai%
r(b,a1)∈CROSS(a2,V)))))∧(SET(opair(b1,d))∧∃a1 b.((b1=a1∧d=b)∧(opair(b,%
a1)∈f∧opair(b,a1)∈CROSS(a2,V)))))⊃c=d) (58)

83 ∀d.((SET(d)∧∃a1 b.(d=opair(a1,b)∧(opair(b,a1)∈f∧opair(b,a1)∈CROSS(%
a2,V))))⊃∃b c.d=opair(b,c))∧∀b1 c d.(((SET(opair(b1,c))∧∃a1 b.((b1=a1%
∧c=b)∧(opair(b,a1)∈f∧opair(b,a1)∈CROSS(a2,V))))∧(SET(opair(b1,d))∧∃a1%
b.((b1=a1∧d=b)∧(opair(b,a1)∈f∧opair(b,a1)∈CROSS(a2,V)))))⊃c=d) (58)

84 (∀d.((SET(d)∧∃a b.(d=opair(a,b)∧opair(b,a)∈f))⊃∃b c.d=opair(b,c))∧%
∀b1 c d.(((SET(opair(b1,c))∧∃a b.((b1=a∧c=b)∧opair(b,a)∈f))∧(SET(opai%
r(b1,d))∧∃a b.((b1=a∧d=b)∧opair(b,a)∈f)))⊃c=d))⊃(∀d.((SET(d)∧∃a1 b.(d%
=opair(a1,b)∧(opair(b,a1)∈f∧opair(b,a1)∈CROSS(a2,V))))⊃∃b c.d=opair(b%
,c))∧∀b1 c d.(((SET(opair(b1,c))∧∃a1 b.((b1=a1∧c=b)∧(opair(b,a1)∈f∧op%
air(b,a1)∈CROSS(a2,V))))∧(SET(opair(b1,d))∧∃a1 b.((b1=a1∧d=b)∧(opair(%
b,a1)∈f∧opair(b,a1)∈CROSS(a2,V)))))⊃c=d))

85 ∀f a2.((∀d.((SET(d)∧∃a b.(d=opair(a,b)∧opair(b,a)∈f))⊃∃b c.d=opair%
(b,c))∧∀b1 c d.(((SET(opair(b1,c))∧∃a b.((b1=a∧c=b)∧opair(b,a)∈f))∧(S%
ET(opair(b1,d))∧∃a b.((b1=a∧d=b)∧opair(b,a)∈f)))⊃c=d))⊃(∀d.((SET(d)∧∃%
a1 b.(d=opair(a1,b)∧(opair(b,a1)∈f∧opair(b,a1)∈CROSS(a2,V))))⊃∃b c.d=%
opair(b,c))∧∀b1 c d.(((SET(opair(b1,c))∧∃a1 b.((b1=a1∧c=b)∧(opair(b,a%
1)∈f∧opair(b,a1)∈CROSS(a2,V))))∧(SET(opair(b1,d))∧∃a1 b.((b1=a1∧d=b)∧%
(opair(b,a1)∈f∧opair(b,a1)∈CROSS(a2,V)))))⊃c=d)))

86 ∀f a.(FNC(CONV(f))⊃FNC(CONV(f | a)))≡∀f a2.((∀d.((SET(d)∧∃a b.(d=o%
pair(a,b)∧opair(b,a)∈f))⊃∃b c.d=opair(b,c))∧∀b1 c d.(((SET(opair(b1,c%
))∧∃a b.((b1=a∧c=b)∧opair(b,a)∈f))∧(SET(opair(b1,d))∧∃a b.((b1=a∧d=b)%

```
∧opair(b,a)∈f)))⊃c=d))⊃(∀d.((SET(d)∧∃a1 b.(d=opair(a1,b)∧(opair(b,a1)%
∈f∧opair(b,a1)∈CROSS(a2,V))))⊃∃b c.d=opair(b,c))∧∀b1 c d.(((SET(opair%
(b1,c))∧∃a1 b.((b1=a1∧c=b)∧(opair(b,a1)∈f∧opair(b,a1)∈CROSS(a2,V))))∧%
(SET(opair(b1,d))∧∃a1 b.((b1=a1∧d=b)∧(opair(b,a1)∈f∧opair(b,a1)∈CROSS%
(a2,V)))))⊃c=d)))

87 ∀f a.(FNC(CONV(f))⊃FNC(CONV(f | a)))

*****
```

.3.4. Domain and range of an one-to-one function.

The next lemma states that the *domain* and the *range* of a *one-to-one* function are *ongruent*. It is proved by a single call to LOGIC, whereas the old proof was in eight ommands.

```
*****GOAL ∀f.(FNC(CONV(f))⊃CONG(DOM(f),RNG(f))) SASSUME CONG;

Goal *4: ∀f.(FNC(CONV(f))⊃CONG(DOM(f),RNG(f)))

*****TRY USING LOGIC;

Goal *4*1: ∀f1.(FNC(CONV(f1))⊃∃f.(FNC(f)∧(FNC(CONV(f))∧(DOM(f)=DOM(f%
1)∧RNG(f)=RNG(f1)))))
Goal *4*1*1: ∀f1.(FNC(CONV(f1))⊃∃f.(FNC(CONV(f))∧(DOM(f)=DOM(f1)∧RNG%
(f)=RNG(f1))))
88 ∀f1.(FNC(CONV(f1))⊃∃f.(FNC(CONV(f))∧(DOM(f)=DOM(f1)∧RNG(f)=RNG(f1)%
)))

89 ∀f1.(FNC(CONV(f1))⊃∃f.(FNC(f)∧(FNC(CONV(f))∧(DOM(f)=DOM(f1)∧RNG(f)%
=RNG(f1)))))≡∀f1.(FNC(CONV(f1))⊃∃f.(FNC(CONV(f))∧(DOM(f)=DOM(f1)∧RNG(%
f)=RNG(f1))))

90 ∀f1.(FNC(CONV(f1))⊃∃f.(FNC(f)∧(FNC(CONV(f))∧(DOM(f)=DOM(f1)∧RNG(f)%
=RNG(f1)))))

91 ∀f.(FNC(CONV(f))⊃CONG(DOM(f),RNG(f)))≡∀f1.(FNC(CONV(f1))⊃∃f.(FNC(f%
)∧(FNC(CONV(f))∧(DOM(f)=DOM(f1)∧RNG(f)=RNG(f1)))))

92 ∀f.(FNC(CONV(f))⊃CONG(DOM(f),RNG(f)))

LOGIC SUCCEEDED!

*****
```

Next we show the FOL proof generated by LOGIC for the above lemma.

```
*****SHOW PROOF 88;;

*****MONADIC ;

88 ∀f1.(FNC(CONV(f1))⊃∃f.(FNC(CONV(f))∧(DOM(f)=DOM(f1)∧RNG(f)=RNG(f1)%
)))

*****SIMPLIFY ∀f1.(FNC(CONV(f1))⊃∃f.(FNC(f)∧(FNC(CONV(f))∧(DOM(f)=DOM%
(f1)∧RNG(f)=RNG(f1)))));

89 ∀f1.(FNC(CONV(f1))⊃∃f.(FNC(f)∧(FNC(CONV(f))∧(DOM(f)=DOM(f1)∧RNG(f)%
=RNG(f1)))))≡∀f1.(FNC(CONV(f1))⊃∃f.(FNC(CONV(f))∧(DOM(f)=DOM(f1)∧RNG(%
f)=RNG(f1))))

*****TAUT ∀f1.(FNC(CONV(f1))⊃∃f.(FNC(f)∧(FNC(CONV(f))∧(DOM(f)=DOM(f1)%
∧RNG(f)=RNG(f1))))) 88,89;

90 ∀f1.(FNC(CONV(f1))⊃∃f.(FNC(f)∧(FNC(CONV(f))∧(DOM(f)=DOM(f1)∧RNG(f)%
=RNG(f1))))

*****REWRITE ∀f.(FNC(CONV(f))⊃CONG(DOM(f),RNG(f))) BY CONG LOGICTRE%
E COMPTREE;

91 ∀f.(FNC(CONV(f))⊃CONG(DOM(f),RNG(f)))≡∀f1.(FNC(CONV(f1))⊃∃f.(FNC(f%
)∧(FNC(CONV(f))∧(DOM(f)=DOM(f1)∧RNG(f)=RNG(f1)))))

*****TAUT ∀f.(FNC(CONV(f))⊃CONG(DOM(f),RNG(f))) 90,91;

92 ∀f.(FNC(CONV(f))⊃CONG(DOM(f),RNG(f)))

*****
```

8.3.5. Range of the restriction.

The last of these lemmas states that *range* of the *restriction* of a function is a subset of the *range* of that function. The old proof was in 23 steps, while the new one takes three steps: two FOL commands followed by a call to LOGIC.

```
*****GOAL ∀f a.RNG(f | a)⊂RNG(f);

Goal #5: ∀f a.RNG(f | a)⊂RNG(f)

*****VE RNG f | a;

93 FNC(f | a)⊃RNG(f | a)={c|∃a1.opair(a1,c)∈(f | a)}
```

```
*****REWRITE ↑ BY {L41}∪LOGICTREE;

2 substitutions were made

94 RNG(f | a)=⁢c|∃a1.opair(a1,c)∈(f | a)⁢}

*****TRY USING LOGIC PLUS SUBSET RNG ↑ RESTR SET AUX25;

Goal *5*1:  ∀f a2 c.((∃b.c∈b∧∃a1.(opair(a1,c)∈f∧opair(a1,c)∈CROSS(a2,%
V)))⊃(∃b.c∈b∧∃a.opair(a,c)∈f))

95 ∀f a2 c.((∃b.c∈b∧∃a1.(opair(a1,c)∈f∧opair(a1,c)∈CROSS(a2,V)))⊃(∃b.%
c∈b∧∃a.opair(a,c)∈f))

96 ∀f a.RNG(f | a)⊂RNG(f)*∀f a2 c.((∃b.c∈b∧∃a1.(opair(a1,c)∈f∧opair(a%
1,c)∈CROSS(a2,V)))⊃(∃b.c∈b∧∃a.opair(a,c)∈f))

97 ∀f a.RNG(f | a)⊂RNG(f)

LOGIC SUCCEEDED!

*****
```

8.4. The GOAL proof of Ramsey's theorem.

We started the proof from scratch. To the axioms listed in the previous sections, we added the last five lemmas as axioms, as follows. The names L41, L95, etc., refer to the line numbers these lemmas had in the old proof.

```
*****DECLARE INDVAR a2 b2 c2 d2 e2;

*****AXIOM L41:∀f a.FNC(f | a);;

L41: ∀f a.FNC(f | a)

*****AXIOM L95:∀f a.(a⊂DOM(f)⊃DOM(f | a)=a);;

L95: ∀f a.(a⊂DOM(f)⊃DOM(f | a)=a)

*****AXIOM L153:∀f a.(FNC(CONV(f))⊃FNC(CONV(f | a)));;

L153: ∀f a.(FNC(CONV(f))⊃FNC(CONV(f | a)))

*****AXIOM L161:∀f.(FNC(CONV(f))⊃CONG(DOM(f),RNG(f)));;

L161: ∀f.(FNC(CONV(f))⊃CONG(DOM(f),RNG(f)))
```

```
*****AXIOM L184:∀f a.RNG(f |a)⊂RNG(f);;

L184: ∀f a.RNG(f | a)⊂RNG(f)

*****
```

The complete proof using GOAL follows. After the proof, we conclude with some statistics about it.

```
*****GOAL ∀G R B.( DENUM(G) ∧ EDGESET(G) = R∪B ∧ R∩B = λ
              ⊃ ∃a.(a⊂G∧DENUM(a)∧(EDGESET(a)⊂B∨EDGESET(a)⊂R)));

Goal #1:  ∀G R B.((DENUM(G)∧(EDGESET(G)=(R∪B)∧(R∩B)=λ))⊃∃a.(a⊂G∧(DENU%
M(a)∧(EDGESET(a)⊂B∨EDGESET(a)⊂R))))

*****TRY USING ∀I;

Goal #1#1:  (DENUM(G)∧(EDGESET(G)=(R∪B)∧(R∩B)=λ))⊃∃a.(a⊂G∧(DENUM(a)∧(%
EDGESET(a)⊂B∨EDGESET(a)⊂R)))

*****TRY USING ⊃I;

Goal #1#1#1:  ∃a.(a⊂G∧(DENUM(a)∧(EDGESET(a)⊂B∨EDGESET(a)⊂R)))

*****PREPARE;

1 DENUM(G)∧(EDGESET(G)=(R∪B)∧(R∩B)=λ)  (1)

2 (R∩B)=λ  (1)

3 EDGESET(G)=(R∪B)  (1)

4 DENUM(G)  (1)

*****LABEL DENUMG ↑;

*****LABEL NOTRB;

*****REWRITE 2 BY {KEXT AUX25 EMPTY}∪LOGICTREE∪COMPTREE;

8 substitutions were made

5 ∀c.¬(c∈R∧c∈B)  (1)

*****LABEL EGETRB;

*****REWRITE 3 BY {KEXT AUX6};

2 substitutions were made
```

```
6 ∀c.(c∈EDGESET(G)≡(c∈R∨c∈B))  (1)

*****LABEL EDGERB ↑;

*****LABEL EDGER;

*****MONADIC ∀c.(c∈R⊃↑:*1*1) ↑;

7 ∀c.(c∈R⊃c∈EDGESET(G))  (1)

*****LABEL EDGEB;

*****MONADIC ∀c.(c∈R⊃↑↑:*1*1) ↑↑;

8 ∀c.(c∈R⊃c∈EDGESET(G))  (1)

*****LABEL SETG;

*****RESOLVE DENUMG AUX1;

RESOLVE DENUM(a)⊃SET(a) , DENUM(G) ↠ SET(G)

9 SET(G)  (1)

*****LABEL NONOG;

*****RESOLVE DENUMG AUX3;

RESOLVE DENUM(a)⊃¬(a=λ) , DENUM(G) ↠ ¬(G=λ)

10 ¬(G=λ)  (1)

*****VE CHOICE G;

11 SET(G)⊃∃f.∀a.((a∈G∧¬(a=λ))⊃(f"a)⊀a)

*****TAUT ↑:*2 SETG ↑;

12 ∃f.∀a.((a∈G∧¬(a=λ))⊃(f"a)⊀a)  (1)

*****LABEL CHOOSEP;

*****ES ↑ p;

13 ∀a.((a∈G∧¬(a=λ))⊃(p"a)⊀a)  (13)

*****VE INDUCTDEF G {b|∃c d.(b=opair(c,d)∧c∈G∧(¬c=λ)
*                    ∧ d=IF DENUM({b|b∈c∧pair(p"c,b)⊀R})
*                          THEN {b|b∈c∧pair(p"c,b)⊀R}
*                          ELSE {b|b∈c∧pair(p"c,b)⊀B})};

14 SET(G)⊃(UNIVERSAL({b|∃c d.(b=opair(c,d)∧(c∈G∧(¬(c=λ)∧d=IF DENUM({b%
|b∈c∧pair(p"c,b)⊀R}) THEN {b|b∈c∧pair(p"c,b)⊀R} ELSE {b|b∈c∧pair(p"c,%
```

```
b)∈B}))))})⊃∃c.∀b.(c=b≡(FNC(b)∧(DOM(b)=omega∧((b"λ)=G∧∀i.(b"SUC(i))=({%
b|∃c d.(b=opair(c,d)∧(c⊂G∧(¬(c=λ)∧d=IF DENUM({b|b∈c∧pair(p"c,b)∈R}) T%
HEN {b|b∈c∧pair(p"c,b)∈R} ELSE {b|b∈c∧pair(p"c,b)∈B})))}"(b"i)))))))) %
*****EVAL ↑;

15 SET(G)⊃∃c.∀b.(c=b≡(FNC(b)∧(DOM(b)=omega∧((b"λ)=G∧∀i.(b"SUC(i))=({b%
|∃c d.(b=opair(c,d)∧(c⊂G∧(¬(c=λ)∧d=IF DENUM({b|b∈c∧pair(p"c,b)∈R}) TH%
EN {b|b∈c∧pair(p"c,b)∈R} ELSE {b|b∈c∧pair(p"c,b)∈B})))}"(b"i))))))

*****REWRITE ↑ BY {SETG}uLOGICTREE;

2 substitutions were made

16 ∃c.∀b.(c=b≡(FNC(b)∧(DOM(b)=omega∧((b"λ)=G∧∀i.(b"SUC(i))=({b|∃c d.(%
b=opair(c,d)∧(c⊂G∧(¬(c=λ)∧d=IF DENUM({b|b∈c∧pair(p"c,b)∈R}) THEN {b|b%
∈c∧pair(p"c,b)∈R} ELSE {b|b∈c∧pair(p"c,b)∈B})))}"(b"i))))))  (1)

*****ES ↑ ee;

17 ∀b.(ee=b≡(FNC(b)∧(DOM(b)=omega∧((b"λ)=G∧∀i.(b"SUC(i))=({b|∃c d.(b=%
opair(c,d)∧(c⊂G∧(¬(c=λ)∧d=IF DENUM({b|b∈c∧pair(p"c,b)∈R}) THEN {b|b∈c%
∧pair(p"c,b)∈R} ELSE {b|b∈c∧pair(p"c,b)∈B})))}"(b"i))))))  (17)

*****∀E ↑ ee;

18 ee=ee≡(FNC(ee)∧(DOM(ee)=omega∧((ee"λ)=G∧∀i.(ee"SUC(i))=({b|∃c d.(b%
=opair(c,d)∧(c⊂G∧(¬(c=λ)∧d=IF DENUM({b|b∈c∧pair(p"c,b)∈R}) THEN {b|b∈%
c∧pair(p"c,b)∈R} ELSE {b|b∈c∧pair(p"c,b)∈B})))}"(ee"i)))))  (17)

*****LABEL IFUNG;

*****REWRITE ↑ BY LOGICTREE;

2 substitutions were made

19 FNC(ee)∧(DOM(ee)=omega∧((ee"λ)=G∧∀i.(ee"SUC(i))=({b|∃c d.(b=opair(%
c,d)∧(c⊂G∧(¬(c=λ)∧d=IF DENUM({b|b∈c∧pair(p"c,b)∈R}) THEN {b|b∈c∧pair(%
p"c,b)∈R} ELSE {b|b∈c∧pair(p"c,b)∈B})))}"(ee"i))))  (17)

*****GOAL ↑:#2#2#2 ASSUME ↑;

Goal #2:  ∀i.(ee"SUC(i))=({b|∃c d.(b=opair(c,d)∧(c⊂G∧(¬(c=λ)∧d=IF DEN%
UM({b|b∈c∧pair(p"c,b)∈R}) THEN {b|b∈c∧pair(p"c,b)∈R} ELSE {b|b∈c∧pair%
(p"c,b)∈B})))}"(ee"i))

*****PREPARE;

20 ∀i.(ee"SUC(i))=({b|∃c d.(b=opair(c,d)∧(c⊂G∧(¬(c=λ)∧d=IF DENUM({b|b%
∈c∧pair(p"c,b)∈R}) THEN {b|b∈c∧pair(p"c,b)∈R} ELSE {b|b∈c∧pair(p"c,b)%
∈B})))}"(ee"i))  (17)

21 (ee"λ)=G  (17)

22 DOM(ee)=omega  (17)
```

```
23 FNC(ee) (17)

*****QED ↑↑↑↑;

*****LABEL EEDEF ↑↑↑↑;

*****LABEL EE0  ↑↑↑;

*****LABEL DOMEE   ↑↑;

*****LABEL FUNEE   ↑;

*****GOAL DENUM({k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})})
*              ∨ DENUM({k|¬DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}) ;

Goal *3:  DENUM({k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})})∨DENUM({k|%
¬DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})})

*****VE AUX10 {k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}
*              {k|¬DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})};

24 UNIVERSAL({k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})})⊃(UNIVERSAL({%
k|¬DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})})⊃(DENUM({k|DENUM({b|b∈(ee"%
k)∧pair(p"(ee"k),b)∈R})}u{k|¬DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})})%
⊃(DENUM({k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})})∨DENUM({k|¬DENUM({%
b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})})))))

*****EVAL ↑;

25 DENUM({k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}u{k|¬DENUM({b|b∈(e%
e"k)∧pair(p"(ee"k),b)∈R})})⊃(DENUM({k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k)%
,b)∈R})})∨DENUM({k|¬DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}))

*****GOAL omega = {k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}
*                    u {k|¬DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})};

Goal *4:  omega=({k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}u{k|¬DENUM%
({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})})

*****SIMPLIFY SET(i);

26 SET(i)

*****LABEL NATSET;

*****∀I ↑ i←a;

27 ∀a.(NATNUM(a)⊃SET(a))

*****TRY USING REWRITE BY {KEXT AUX6 omega};

Goal *4*1:  ∀c.((SET(c)∧NATNUM(c))⊃((NATNUM(c)∧DENUM({b|b∈(ee"c)∧pair%
(p"(ee"c),b)∈R}))∨(NATNUM(c)∧¬DENUM({b|b∈(ee"c)∧pair(p"(ee"c),b)∈R}))))
```

```
*****TRY USING MONADIC NATSET;

28 ∀c.((SET(c)∧NATNUM(c))≡((NATNUM(c)∧DENUM({b|b∈(ee"c)∧pair(p"(ee"c)%
,b)∈R}))∨(NATNUM(c)∧¬DENUM({b|b∈(ee"c)∧pair(p"(ee"c),b)∈R}))))

29 omega=({k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}∪{k|¬DENUM({b|b∈(%
ee"k)∧pair(p"(ee"k),b)∈R})})≡∀c.((SET(c)∧NATNUM(c))≡((NATNUM(c)∧DENUM%
({b|b∈(ee"c)∧pair(p"(ee"c),b)∈R}))∨(NATNUM(c)∧¬DENUM({b|b∈(ee"c)∧pair%
(p"(ee"c),b)∈R}))))

30 omega=({k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}∪{k|¬DENUM({b|b∈(%
ee"k)∧pair(p"(ee"k),b)∈R})})


*****TRY #3 USING MONADIC ↑,↑↑↑↑↑↑ AUX30;

31 DENUM({k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})})∨DENUM({k|¬DENUM(%
{b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})})


*****TRY #1#1#1 USING EG  RNG({b|∃k.b=opair(k,p"(ee"k))} |
*                      IF DENUM({k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})})
*                        THEN {k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}
*                        ELSE {k|¬DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}) ;

Goal #1#1#1#1:  RNG({b|∃k.b=opair(k,p"(ee"k))} | IF DENUM({k|DENUM({b%
|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}) THEN {k|DENUM({b|b∈(ee"k)∧pair(p"(ee%
"k),b)∈R})} ELSE {k|¬DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})})⊂G∧(DENU%
M(RNG({b|∃k.b=opair(k,p"(ee"k))} | IF DENUM({k|DENUM({b|b∈(ee"k)∧pair%
(p"(ee"k),b)∈R})}) THEN {k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})} EL%
SE {k|¬DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}))∧(EDGESET(RNG({b|∃k.b%
=opair(k,p"(ee"k))} | IF DENUM({k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈%
R})}) THEN {k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})} ELSE {k|¬DENUM(%
{b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}))⊂B∨EDGESET(RNG({b|∃k.b=opair(k,p"(%
ee"k))} | IF DENUM({k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}) THEN {%
k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})} ELSE {k|¬DENUM({b|b∈(ee"k)∧%
pair(p"(ee"k),b)∈R})}))⊂R))

*****GOAL FNC({b|∃k.b=opair(k,p"(ee"k))});

Goal #5:  FNC({b|∃k.b=opair(k,p"(ee"k))})

*****TRY USING REWRITE BY {FNC REL AUX27};

Goal #5#1:  ∀d.((SET(d)∧∃k.d=opair(k,p"(ee"k)))⊃∃b c.d=opair(b,c))∧∀b%
 c d.(((SET(opair(b,c))∧∃k.(b=k∧c=(p"(ee"k))))∧(SET(opair(b,d))∧∃k.(b%
=k∧d=(p"(ee"k)))))⊃c=d)

*****TRY USING ELIMINATION DEPTH 2;

Goal #5#1#1:  ∀d.((SET(d)∧∃k.d=opair(k,p"(ee"k)))⊃∃b c.d=opair(b,c))
Goal #5#1#2:  ∀b c d.(((SET(opair(b,c))∧∃k.(b=k∧c=(p"(ee"k))))∧(SET(o%
pair(b,d))∧∃k.(b=k∧d=(p"(ee"k)))))⊃c=d)
Goal #5#1#1#1:  (SET(d)∧∃k.d=opair(k,p"(ee"k)))⊃∃b c.d=opair(b,c)
```

```
Goal #5#1#2#1: ((SET(opair(b,c))∧∃k.(b=k∧c=(p"(ee"k))))∧(SET(opair(b%
,d))∧∃k.(b=k∧d=(p"(ee"k)))))⊃c=d

*****TRY #5#1#1#1 USING LOGIC;


32 (SET(d)∧∃k.d=opair(k,p"(ee"k)))⊃∃b c.d=opair(b,c)

33 ∀d.((SET(d)∧∃k.d=opair(k,p"(ee"k)))⊃∃b c.d=opair(b,c))

Goal #5#1#2#1#1: c=d

34 (SET(opair(b,c))∧∃k.(b=k∧c=(p"(ee"k))))∧(SET(opair(b,d))∧∃k.(b=k∧d%
=(p"(ee"k)))) (34)

35 ∃k.(b=k∧d=(p"(ee"k))) (34)

36 SET(opair(b,d)) (34)

37 ∃k.(b=k∧c=(p"(ee"k))) (34)

38 SET(opair(b,c)) (34)

We have a failqueue of length: 1
Starting a new 2-th pass on new queue of length: 1
We have a failqueue of length: 1
Failure: can't prove anything on failqueue.
The tactic LOGIC can't be applied to goal
Goal #5#1#1#1: Proven. 32 (SET(d)∧∃k.d=opair(k,p"(ee"k)))⊃∃b c.d%
=opair(b,c)

*****ES ↑↑↑↑ k;

39 b=k∧d=(p"(ee"k)) (39)

*****ES ↑↑↑ j;

40 b=j∧c=(p"(ee"j)) (40)

*****TAUTEQ k=j ↑↑;;

41 k=j (39 40)

*****REWRITE ↑↑↑ BY {↑};

2 substitutions were made

42 b=j∧d=(p"(ee"j)) (34 40)

*****TRY USING TAUTEQ ↑↑↑,↑;

43 c=d (34)
```

```
44 ((SET(opair(b,c))∧∃k.(b=k∧c=(p"(ee"k))))∧(SET(opair(b,d))∧∃k.(b=k∧%
d=(p"(ee"k)))))⊃c=d

45 ∀b c d.(((SET(opair(b,c))∧∃k.(b=k∧c=(p"(ee"k))))∧(SET(opair(b,d))∧%
∃k.(b=k∧d=(p"(ee"k)))))⊃c=d)

46 ∀d.((SET(d)∧∃k.d=opair(k,p"(ee"k)))⊃∃b c.d=opair(b,c))∧∀b c d.(((S%
ET(opair(b,c))∧∃k.(b=k∧c=(p"(ee"k))))∧(SET(opair(b,d))∧∃k.(b=k∧d=(p"(%
ee"k)))))⊃c=d)

47 FNC({b|∃k.b=opair(k,p"(ee"k))})≡(∀d.((SET(d)∧∃k.d=opair(k,p"(ee"k)%
))⊃∃b c.d=opair(b,c))∧∀b c d.(((SET(opair(b,c))∧∃k.(b=k∧c=(p"(ee"k)))%
)∧(SET(opair(b,d))∧∃k.(b=k∧d=(p"(ee"k)))))⊃c=d))

48 FNC({b|∃k.b=opair(k,p"(ee"k))})


*****LABEL FUNCC ↑;

*****VE AUX10 {b|b∈c∧pair(a,b)∢R} {b|b∈c∧pair(a,b)∢B};

49 UNIVERSAL({b|b∈c∧pair(a,b)∢R})⊃(UNIVERSAL({b|b∈c∧pair(a,b)∢B})⊃(DE%
NUM({b|b∈c∧pair(a,b)∢R}∪{b|b∈c∧pair(a,b)∢B})⊃(DENUM({b|b∈c∧pair(a,b)∢%
R})∨DENUM({b|b∈c∧pair(a,b)∢B}))))

*****EVAL ↑;

50 DENUM({b|b∈c∧pair(a,b)∢R}∪{b|b∈c∧pair(a,b)∢B})⊃(DENUM({b|b∈c∧pair(%
a,b)∢R})∨DENUM({b|b∈c∧pair(a,b)∢B}))

*****GOAL ∀c a.(c⊂G∧a∈c⊃↑:*1*1=c\singl(a));

Goal *6: ∀c a.((c⊂G∧a∈c)⊃({b|b∈c∧pair(a,b)∢R}∪{b|b∈c∧pair(a,b)∢B})=(%
c \ singl(a)))

*****TRY USING REWRITE BY {KEXT SUBSET AUX6};

Goal *6*1: ∀c1 a.((∀c.(c∈c1⊃c∈G)∧a∈c1)⊃∀c.(((SET(c)∧(c∈c1∧pair(a,c)∢%
R))∨(SET(c)∧(c∈c1∧pair(a,c)∢B)))≡c∈(c1 \ singl(a))))

*****TRY USING REWRITE BY {DIFF COMPL  AUX25 UNIT V};

Goal *6*1*1: ∀c1 a.((∀c.(c∈c1⊃c∈G)∧a∈c1)⊃∀c.(((SET(c)∧(c∈c1∧pair(a,c%
)∢R))∨(SET(c)∧(c∈c1∧pair(a,c)∢B)))≡(c∈c1∧(SET(c)∧¬(SET(c)∧(SET(a)∧c=a%
))))))

*****TRY USING ELIMINATION DEPTH 3;

Goal *6*1*1*1: (∀c.(c∈c1⊃c∈G)∧a∈c1)⊃∀c.(((SET(c)∧(c∈c1∧pair(a,c)∢R))%
∨(SET(c)∧(c∈c1∧pair(a,c)∢B)))≡(c∈c1∧(SET(c)∧¬(SET(c)∧(SET(a)∧c=a)))))
Goal *6*1*1*1*1: ∀c.(((SET(c)∧(c∈c1∧pair(a,c)∢R))∨(SET(c)∧(c∈c1∧pair%
(a,c)∢B)))≡(c∈c1∧(SET(c)∧¬(SET(c)∧(SET(a)∧c=a)))))
51 ∀c.(c∈c1⊃c∈G)∧a∈c1  (51)
```

52 a∈c1 (51)

53 ∀c.(c∈c1⊃c∈G) (51)

Goal *6*1*1*1*1*1: ((SET(c)∧(c∈c1∧pair(a,c)∈R))∨(SET(c)∧(c∈c1∧pair(a%
,c)∈B))≡(c∈c1∧(SET(c)∧¬(SET(c)∧(SET(a)∧c=a))))

*****∀E AUX12 a c G;

54 pair(a,c)∈EDGESET(G)≡(a∈G∧(c∈G∧¬(a=c)))

*****REWRITE ↑ BY {3 AUX6};

2 substitutions were made

55 (pair(a,c)∈R∨pair(a,c)∈B)≡(a∈G∧(c∈G∧¬(a=c))) (1)

*****TRY USING MONADIC ↑↑↑↑,↑↑↑,↑;

56 ((SET(c)∧(c∈c1∧pair(a,c)∈R))∨(SET(c)∧(c∈c1∧pair(a,c)∈B)))≡(c∈c1∧(S%
ET(c)∧¬(SET(c)∧(SET(a)∧c=a)))) (1 51)

57 ∀c.(((SET(c)∧(c∈c1∧pair(a,c)∈R))∨(SET(c)∧(c∈c1∧pair(a,c)∈B)))≡(c∈c%
1∧(SET(c)∧¬(SET(c)∧(SET(a)∧c=a))))) (1 51)

58 (∀c.(c∈c1⊃c∈G)∧a∈c1)⊃∀c.(((SET(c)∧(c∈c1∧pair(a,c)∈R))∨(SET(c)∧(c∈c%
1∧pair(a,c)∈B)))≡(c∈c1∧(SET(c)∧¬(SET(c)∧(SET(a)∧c=a))))) (1)

59 ∀c1 a.((∀c.(c∈c1⊃c∈G)∧a∈c1)⊃∀c.(((SET(c)∧(c∈c1∧pair(a,c)∈R))∨(SET(%
c)∧(c∈c1∧pair(a,c)∈B)))≡(c∈c1∧(SET(c)∧¬(SET(c)∧(SET(a)∧c=a)))))) (1)

60 ∀c1 a.((∀c.(c∈c1⊃c∈G)∧a∈c1)⊃∀c.(((SET(c)∧(c∈c1∧pair(a,c)∈R))∨(SET(%
c)∧(c∈c1∧pair(a,c)∈B)))≡c∈(c1 \ singl(a))))≡∀c1 a.((∀c.(c∈c1⊃c∈G)∧a∈c%
1)⊃∀c.(((SET(c)∧(c∈c1∧pair(a,c)∈R))∨(SET(c)∧(c∈c1∧pair(a,c)∈B)))≡(c∈c%
1∧(SET(c)∧¬(SET(c)∧(SET(a)∧c=a))))))

61 ∀c1 a.((∀c.(c∈c1⊃c∈G)∧a∈c1)⊃∀c.(((SET(c)∧(c∈c1∧pair(a,c)∈R))∨(SET(%
c)∧(c∈c1∧pair(a,c)∈B)))≡c∈(c1 \ singl(a)))) (1)

62 ∀c a.((c⊂G∧a∈c)⊃({b|b∈c∧pair(a,b)∈R}∪{b|b∈c∧pair(a,b)∈B})=(c \ sin%
gl(a)))≡∀c1 a.((∀c.(c∈c1⊃c∈G)∧a∈c1)⊃∀c.(((SET(c)∧(c∈c1∧pair(a,c)∈R))∨%
(SET(c)∧(c∈c1∧pair(a,c)∈B)))≡c∈(c1 \ singl(a))))

63 ∀c a.((c⊂G∧a∈c)⊃({b|b∈c∧pair(a,b)∈R}∪{b|b∈c∧pair(a,b)∈B})=(c \ sin%
gl(a))) (1)

*****GOAL ∀c a.(c⊂G∧DENUM(c)∧a∈c⊃50:≡2);

Goal *7: ∀c a.((c⊂G∧(DENUM(c)∧a∈c))⊃(DENUM({b|b∈c∧pair(a,b)∈R})∨DENU%
M({b|b∈c∧pair(a,b)∈B})))

*****TRY USING ELIMINATION;

```
Goal *7*1: (c⊂G∧(DENUM(c)∧a∈c))⊃(DENUM({b|b∈c∧pair(a,b)∈R})∨DENUM({b%
|b∈c∧pair(a,b)∈B}))
Goal *7*1*1: DENUM({b|b∈c∧pair(a,b)∈R})∨DENUM({b|b∈c∧pair(a,b)∈B})

*****TRY USING IMPLICATION 50;

64 c⊂G∧(DENUM(c)∧a∈c) (64)

65 a∈c (64)

66 DENUM(c) (64)

67 c⊂G (64)

Goal *7*1*1*1: DENUM({b|b∈c∧pair(a,b)∈R}∪{b|b∈c∧pair(a,b)∈B})

*****∀E 63 c a;

68 (c⊂G∧a∈c)⊃({b|b∈c∧pair(a,b)∈R}∪{b|b∈c∧pair(a,b)∈B})=(c \ singl(a)) (1)

*****TAUT ↑:*2 64 ↑;

69 ({b|b∈c∧pair(a,b)∈R}∪{b|b∈c∧pair(a,b)∈B})=(c \ singl(a)) (1 64)

*****RESOLVE 66 AUX9;

RESOLVE DENUM(a)⊃DENUM(a \ singl(b)) , DENUM(c) →→ ∀b.DENUM(c \ singl(b))

70 ∀b.DENUM(c \ singl(b)) (64)

*****TRY USING REWRITE BY {↑↑,↑};

71 DENUM({b|b∈c∧pair(a,b)∈R}∪{b|b∈c∧pair(a,b)∈B}) (1 64)

RESOLVE DENUM({b|b∈c∧pair(a,b)∈R}∪{b|b∈c∧pair(a,b)∈B})⊃(DENUM({b|b∈c∧%
pair(a,b)∈R})∨DENUM({b|b∈c∧pair(a,b)∈B})) , DENUM({b|b∈c∧pair(a,b)∈R}%
∪{b|b∈c∧pair(a,b)∈B}) →→ DENUM({b|b∈c∧pair(a,b)∈R})∨DENUM({b|b∈c∧pair%
(a,b)∈B})

72 DENUM({b|b∈c∧pair(a,b)∈R})∨DENUM({b|b∈c∧pair(a,b)∈B}) (1 64)

73 (c⊂G∧(DENUM(c)∧a∈c))⊃(DENUM({b|b∈c∧pair(a,b)∈R})∨DENUM({b|b∈c∧pair%
(a,b)∈B})) (1)

74 ∀c a.((c⊂G∧(DENUM(c)∧a∈c))⊃(DENUM({b|b∈c∧pair(a,b)∈R})∨DENUM({b|b∈%
c∧pair(a,b)∈B}))) (1)

*****GOAL ∀i.(ee"i⊂G∧DENUM(ee"i));

Goal *8: ∀i.((ee"i)⊂G∧DENUM(ee"i))

*****TRY USING INDUCTION;
```

```
Goal *8*1: (ee"λ)⊂G∧DENUM(ee"λ)
Goal *8*2: ∀i.(((ee"i)⊂G∧DENUM(ee"i))⊃((ee"SUC(i))⊂G∧DENUM(ee"SUC(i))))

*****TRY 1 USING REWRITE BY {SUBSET EE0 DENUMG};

75 (ee"λ)⊂G∧DENUM(ee"λ) (1 17)


*****TRY USING ELIMINATION;

76 DENUM(ee"λ) (1 17)

77 (ee"λ)⊂G (1 17)

Goal *8*2*1: ((ee"i)⊂G∧DENUM(ee"i))⊃((ee"SUC(i))⊂G∧DENUM(ee"SUC(i)))
Goal *8*2*1*1: (ee"SUC(i))⊂G∧DENUM(ee"SUC(i))
78 (ee"i)⊂G∧DENUM(ee"i) (78)

79 DENUM(ee"i) (78)

80 (ee"i)⊂G (78)

Goal *8*2*1*1*1: (ee"SUC(i))⊂G
Goal *8*2*1*1*2: DENUM(ee"SUC(i))

*****∀E APPLY EEDEF:*1*2*1 ee"i;

81 UNIVERSAL({b|∃c d.(b=opair(c,d)∧(c⊂G∧(¬(c=λ)∧d=IF DENUM({b|b∈c∧pai%
r(p"c,b)∈R}) THEN {b|b∈c∧pair(p"c,b)∈R} ELSE {b|b∈c∧pair(p"c,b)∈B}))))%
})⊃((∃d.∀c.(d=c≠opair(ee"i,c)∈{b|∃c d.(b=opair(c,d)∧(c⊂G∧(¬(c=λ)∧d=IF%
 DENUM({b|b∈c∧pair(p"c,b)∈R}) THEN {b|b∈c∧pair(p"c,b)∈R} ELSE {b|b∈c∧%
pair(p"c,b)∈B}))))})⊃opair(ee"i,{b|∃c d.(b=opair(c,d)∧(c⊂G∧(¬(c=λ)∧d=I%
F DENUM({b|b∈c∧pair(p"c,b)∈R}) THEN {b|b∈c∧pair(p"c,b)∈R} ELSE {b|b∈c%
∧pair(p"c,b)∈B}))))}"(ee"i))∈{b|∃c d.(b=opair(c,d)∧(c⊂G∧(¬(c=λ)∧d=IF D%
ENUM({b|b∈c∧pair(p"c,b)∈R}) THEN {b|b∈c∧pair(p"c,b)∈R} ELSE {b|b∈c∧pa%
ir(p"c,b)∈B}))))})∧(¬∃d.∀c.(d=c≠opair(ee"i,c)∈{b|∃c d.(b=opair(c,d)∧(c%
⊂G∧(¬(c=λ)∧d=IF DENUM({b|b∈c∧pair(p"c,b)∈R}) THEN {b|b∈c∧pair(p"c,b)∈%
R} ELSE {b|b∈c∧pair(p"c,b)∈B}))))})⊃({b|∃c d.(b=opair(c,d)∧(c⊂G∧(¬(c=λ%
)∧d=IF DENUM({b|b∈c∧pair(p"c,b)∈R}) THEN {b|b∈c∧pair(p"c,b)∈R} ELSE {%
b|b∈c∧pair(p"c,b)∈B}))))}"(ee"i))=λ))

*****EVAL ↑;

82 (∃d.∀c.(d=c≠opair(ee"i,c)∈{b|∃c d.(b=opair(c,d)∧(c⊂G∧(¬(c=λ)∧d=IF %
DENUM({b|b∈c∧pair(p"c,b)∈R}) THEN {b|b∈c∧pair(p"c,b)∈R} ELSE {b|b∈c∧p%
air(p"c,b)∈B}))))})⊃opair(ee"i,{b|∃c d.(b=opair(c,d)∧(c⊂G∧(¬(c=λ)∧d=IF%
 DENUM({b|b∈c∧pair(p"c,b)∈R}) THEN {b|b∈c∧pair(p"c,b)∈R} ELSE {b|b∈c∧%
pair(p"c,b)∈B}))))}"(ee"i))∈{b|∃c d.(b=opair(c,d)∧(c⊂G∧(¬(c=λ)∧d=IF DE%
NUM({b|b∈c∧pair(p"c,b)∈R}) THEN {b|b∈c∧pair(p"c,b)∈R} ELSE {b|b∈c∧pai%
r(p"c,b)∈B}))))})∧(¬∃d.∀c.(d=c≠opair(ee"i,c)∈{b|∃c d.(b=opair(c,d)∧(c⊂%
G∧(¬(c=λ)∧d=IF DENUM({b|b∈c∧pair(p"c,b)∈R}) THEN {b|b∈c∧pair(p"c,b)∈R%
} ELSE {b|b∈c∧pair(p"c,b)∈B}))))})⊃({b|∃c d.(b=opair(c,d)∧(c⊂G∧(¬(c=λ)%
∧d=IF DENUM({b|b∈c∧pair(p"c,b)∈R}) THEN {b|b∈c∧pair(p"c,b)∈R} ELSE {b%
|b∈c∧pair(p"c,b)∈B}))))}"(ee"i))=λ)
```

```
*****∧E ↑ 1;

83 ∃d.∀c.(d=c≈opair(ee"i,c)∈{b|∃c d.(b=opair(c,d)∧(c⊂G∧(¬(c=λ)∧d=IF D%
ENUM({b|b∈c∧pair(p"c,b)∈R}) THEN {b|b∈c∧pair(p"c,b)∈R} ELSE {b|b∈c∧pa%
ir(p"c,b)∈B}))))}⊃opair(ee"i,{b|∃c d.(b=opair(c,d)∧(c⊂G∧(¬(c=λ)∧d=IF %
DENUM({b|b∈c∧pair(p"c,b)∈R}) THEN {b|b∈c∧pair(p"c,b)∈R} ELSE {b|b∈c∧p%
air(p"c,b)∈B}))))}"(ee"i))∈{b|∃c d.(b=opair(c,d)∧(c⊂G∧(¬(c=λ)∧d=IF DEN%
UM({b|b∈c∧pair(p"c,b)∈R}) THEN {b|b∈c∧pair(p"c,b)∈R} ELSE {b|b∈c∧pair%
(p"c,b)∈B}))))}

*****GOAL ↑:≈2 ASSUME ↑;

Goal ≈9: opair(ee"i,{b|∃c d.(b=opair(c,d)∧(c⊂G∧(¬(c=λ)∧d=IF DENUM({b%
|b∈c∧pair(p"c,b)∈R}) THEN {b|b∈c∧pair(p"c,b)∈R} ELSE {b|b∈c∧pair(p"c,%
b)∈B}))))}"(ee"i))∈{b|∃c d.(b=opair(c,d)∧(c⊂G∧(¬(c=λ)∧d=IF DENUM({b|b∈c%
∧pair(p"c,b)∈R}) THEN {b|b∈c∧pair(p"c,b)∈R} ELSE {b|b∈c∧pair(p"c,b)∈B}))))}

*****TRY USING IMPLICATION;


Goal ≈9≈1: ∃d.∀c.(d=c≈opair(ee"i,c)∈{b|∃c d.(b=opair(c,d)∧(c⊂G∧(¬(c=%
λ)∧d=IF DENUM({b|b∈c∧pair(p"c,b)∈R}) THEN {b|b∈c∧pair(p"c,b)∈R} ELSE %
{b|b∈c∧pair(p"c,b)∈B}))))})

*****TRY USING EG IF DENUM({b|b∈ee"i∧pair(p"(ee"i),b)∈R})
* THEN {b|b∈ee"i∧pair(p"(ee"i),b)∈R}
* ELSE {b|b∈ee"i∧pair(p"(ee"i),b)∈B};


Goal ≈9≈1≈1: ∀c.(IF DENUM({b|b∈(ee"i)∧pair(p"(ee"i),b)∈R}) THEN {b|b%
∈(ee"i)∧pair(p"(ee"i),b)∈R} ELSE {b|b∈(ee"i)∧pair(p"(ee"i),b)∈B}=c≈op%
air(ee"i,c)∈{b|∃c d.(b=opair(c,d)∧(c⊂G∧(¬(c=λ)∧d=IF DENUM({b|b∈c∧pair%
(p"c,b)∈R}) THEN {b|b∈c∧pair(p"c,b)∈R} ELSE {b|b∈c∧pair(p"c,b)∈B})))))

*****TRY USING REWRITE BY {AUX5 AUX27};

Goal ≈9≈1≈1≈1: ∀c.(IF DENUM({b|b∈(ee"i)∧pair(p"(ee"i),b)∈R}) THEN {b%
|b∈(ee"i)∧pair(p"(ee"i),b)∈R} ELSE {b|b∈(ee"i)∧pair(p"(ee"i),b)∈B}=c≈%
((SET(ee"i)∧SET(c))∧∃c1 d.(((ee"i)=c1∧c=d)∧(c1⊂G∧(¬(c1=λ)∧d=IF DENUM(%
{b|b∈c1∧pair(p"c1,b)∈R}) THEN {b|b∈c1∧pair(p"c1,b)∈R} ELSE {b|b∈c1∧pa%
ir(p"c1,b)∈B}))))))

*****TRY USING ELIMINATION DEPTH 3;

Goal ≈9≈1≈1≈1≈1: IF DENUM({b|b∈(ee"i)∧pair(p"(ee"i),b)∈R}) THEN {b|b%
∈(ee"i)∧pair(p"(ee"i),b)∈R} ELSE {b|b∈(ee"i)∧pair(p"(ee"i),b)∈B}=c≈((%
SET(ee"i)∧SET(c))∧∃c1 d.(((ee"i)=c1∧c=d)∧(c1⊂G∧(¬(c1=λ)∧d=IF DENUM({b%
|b∈c1∧pair(p"c1,b)∈R}) THEN {b|b∈c1∧pair(p"c1,b)∈R} ELSE {b|b∈c1∧pair%
(p"c1,b)∈B})))))
Goal ≈9≈1≈1≈1≈1≈1: IF DENUM({b|b∈(ee"i)∧pair(p"(ee"i),b)∈R}) THEN {b%
|b∈(ee"i)∧pair(p"(ee"i),b)∈R} ELSE {b|b∈(ee"i)∧pair(p"(ee"i),b)∈B}=c⊃%
((SET(ee"i)∧SET(c))∧∃c1 d.(((ee"i)=c1∧c=d)∧(c1⊂G∧(¬(c1=λ)∧d=IF DENUM(%
{b|b∈c1∧pair(p"c1,b)∈R}) THEN {b|b∈c1∧pair(p"c1,b)∈R} ELSE {b|b∈c1∧pa%
ir(p"c1,b)∈B})))))
```

```
Goal #9*1*1*1*1*2: ((SET(ee"i)∧SET(c))∧∃c1 d.(((ee"i)=c1∧c=d)∧(c1⊂G∧%
(¬(c1=λ)∧d=IF DENUM({b|b∈c1∧pair(p"c1,b)∈R}) THEN {b|b∈c1∧pair(p"c1,b%
)∈R} ELSE {b|b∈c1∧pair(p"c1,b)∈B})))⊃IF DENUM({b|b∈(ee"i)∧pair(p"(ee%
"i),b)∈R}) THEN {b|b∈(ee"i)∧pair(p"(ee"i),b)∈R} ELSE {b|b∈(ee"i)∧pair%
(p"(ee"i),b)∈B}=c
Goal #9*1*1*1*1*1*1: (SET(ee"i)∧SET(c))∧∃c1 d.(((ee"i)=c1∧c=d)∧(c1⊂G%
∧(¬(c1=λ)∧d=IF DENUM({b|b∈c1∧pair(p"c1,b)∈R}) THEN {b|b∈c1∧pair(p"c1,%
b)∈R} ELSE {b|b∈c1∧pair(p"c1,b)∈B})))
Goal #9*1*1*1*1*2*1: IF DENUM({b|b∈(ee"i)∧pair(p"(ee"i),b)∈R}) THEN %
{b|b∈(ee"i)∧pair(p"(ee"i),b)∈R} ELSE {b|b∈(ee"i)∧pair(p"(ee"i),b)∈B}=c

*****PREPARE;

84 (SET(ee"i)∧SET(c))∧∃c1 d.(((ee"i)=c1∧c=d)∧(c1⊂G∧(¬(c1=λ)∧d=IF DENU%
M({b|b∈c1∧pair(p"c1,b)∈R}) THEN {b|b∈c1∧pair(p"c1,b)∈R} ELSE {b|b∈c1∧%
pair(p"c1,b)∈B}))) (84)

85 ∃c1 d.(((ee"i)=c1∧c=d)∧(c1⊂G∧(¬(c1=λ)∧d=IF DENUM({b|b∈c1∧pair(p"c1%
,b)∈R}) THEN {b|b∈c1∧pair(p"c1,b)∈R} ELSE {b|b∈c1∧pair(p"c1,b)∈B}))) (84)

86 SET(c) (84)

87 SET(ee"i) (84)

*****ES ↑↑↑ c1 d;

88 ((ee"i)=c1∧c=d)∧(c1⊂G∧(¬(c1=λ)∧d=IF DENUM({b|b∈c1∧pair(p"c1,b)∈R})%
 THEN {b|b∈c1∧pair(p"c1,b)∈R} ELSE {b|b∈c1∧pair(p"c1,b)∈B})) (88)

*****ADDFACTS #9*1*1*1*1*2*1 ASSUME ↑;

Goal #9*1*1*1*1*2*1: IF DENUM({b|b∈(ee"i)∧pair(p"(ee"i),b)∈R}) THEN %
{b|b∈(ee"i)∧pair(p"(ee"i),b)∈R} ELSE {b|b∈(ee"i)∧pair(p"(ee"i),b)∈B}=c

*****PREPARE;

89 d=IF DENUM({b|b∈c1∧pair(p"c1,b)∈R}) THEN {b|b∈c1∧pair(p"c1,b)∈R} E%
LSE {b|b∈c1∧pair(p"c1,b)∈B} (88)

90 ¬(c1=λ) (88)

91 c1⊂G (88)

92 c=d (88)

93 (ee"i)=c1 (88)

94 c1=λ≡FALSE (88)

*****TRY USING REWRITE BY {↑↑,↑↑↑,↑↑↑↑↑↑};

95 IF DENUM({b|b∈(ee"i)∧pair(p"(ee"i),b)∈R}) THEN {b|b∈(ee"i)∧pair(p"%
(ee"i),b)∈R} ELSE {b|b∈(ee"i)∧pair(p"(ee"i),b)∈B}=c (84)
```

```
96 ((SET(ee"i)∧SET(c))∧∃c1 d.(((ee"i)=c1∧c=d)∧(c1⊂G∧(¬(c1=λ)∧d=IF DEN%
UM({b|b∈c1∧pair(p"c1,b)∈R}) THEN {b|b∈c1∧pair(p"c1,b)∈R} ELSE {b|b∈c1%
∧pair(p"c1,b)∈B})))⊃IF DENUM({b|b∈(ee"i)∧pair(p"(ee"i),b)∈R}) THEN {%
b|b∈(ee"i)∧pair(p"(ee"i),b)∈R} ELSE {b|b∈(ee"i)∧pair(p"(ee"i),b)∈B}=c%

*****TRY USING ∧I;

97 IF DENUM({b|b∈(ee"i)∧pair(p"(ee"i),b)∈R}) THEN {b|b∈(ee"i)∧pair(p"%
(ee"i),b)∈R} ELSE {b|b∈(ee"i)∧pair(p"(ee"i),b)∈B}=c  (97)

Goal #9#1#1#1#1#1#1#1: SET(ee"i)∧SET(c)
Goal #9#1#1#1#1#1#1#2: ∃c1 d.(((ee"i)=c1∧c=d)∧(c1⊂G∧(¬(c1=λ)∧d=IF DE%
NUM({b|b∈c1∧pair(p"c1,b)∈R}) THEN {b|b∈c1∧pair(p"c1,b)∈R} ELSE {b|b∈c%
1∧pair(p"c1,b)∈B})))

*****TRY USING EG ee"i ↑:#1;

Goal #9#1#1#1#1#1#1#2#1: ((ee"i)=(ee"i)∧c=IF DENUM({b|b∈(ee"i)∧pair(%
p"(ee"i),b)∈R}) THEN {b|b∈(ee"i)∧pair(p"(ee"i),b)∈R} ELSE {b|b∈(ee"i)%
∧pair(p"(ee"i),b)∈B})∧((ee"i)⊂G∧(¬((ee"i)=λ)∧IF DENUM({b|b∈(ee"i)∧pai%
r(p"(ee"i),b)∈R}) THEN {b|b∈(ee"i)∧pair(p"(ee"i),b)∈R} ELSE {b|b∈(ee"%
i)∧pair(p"(ee"i),b)∈B}=IF DENUM({b|b∈(ee"i)∧pair(p"(ee"i),b)∈R}) THEN%
 {b|b∈(ee"i)∧pair(p"(ee"i),b)∈R} ELSE {b|b∈(ee"i)∧pair(p"(ee"i),b)∈B}))

*****∀E AUX3 ee"i;

98 DENUM(ee"i)⊃¬((ee"i)=λ)

*****TAUT ↑:#2 79 ↑;

99 ¬((ee"i)=λ)  (78)

*****TRY USING REWRITE BY {80 ↑,↑↑↑};

100 ((ee"i)=(ee"i)∧c=IF DENUM({b|b∈(ee"i)∧pair(p"(ee"i),b)∈R}) THEN {%
b|b∈(ee"i)∧pair(p"(ee"i),b)∈R} ELSE {b|b∈(ee"i)∧pair(p"(ee"i),b)∈B})∧%
((ee"i)⊂G∧(¬((ee"i)=λ)∧IF DENUM({b|b∈(ee"i)∧pair(p"(ee"i),b)∈R}) THEN%
 {b|b∈(ee"i)∧pair(p"(ee"i),b)∈R} ELSE {b|b∈(ee"i)∧pair(p"(ee"i),b)∈B}%
=IF DENUM({b|b∈(ee"i)∧pair(p"(ee"i),b)∈R}) THEN {b|b∈(ee"i)∧pair(p"(e%
e"i),b)∈R} ELSE {b|b∈(ee"i)∧pair(p"(ee"i),b)∈B}))  (78 97)

101 ∃c1 d.(((ee"i)=c1∧c=d)∧(c1⊂G∧(¬(c1=λ)∧d=IF DENUM({b|b∈c1∧pair(p"c1,b)%
∈R}) THEN {b|b∈c1∧pair(p"c1,b)∈R} ELSE {b|b∈c1∧pair(p"c1,b)∈B})))  (78 97)


*****∀E 74 ee"i p"(ee"i);

102 ((ee"i)⊂G∧(DENUM(ee"i)∧(p"(ee"i))∈(ee"i)))⊃(DENUM({b|b∈(ee"i)∧pai%
r(p"(ee"i),b)∈R})∨DENUM({b|b∈(ee"i)∧pair(p"(ee"i),b)∈B}))  (1)

*****∀E CHOOSEP ee"i;

103 ((ee"i)⊂G∧¬(ee"i)=λ))⊃(p"(ee"i))∈(ee"i)  (13)
```

```
*****TAUT DENUM(97:#1) 79 80 99 ↑↑;;

104 DENUM(IF DENUM({b|b∈(ee"i)∧pair(p"(ee"i),b)∈R}) THEN {b|b∈(ee"i)∧%
pair(p"(ee"i),b)∈R} ELSE {b|b∈(ee"i)∧pair(p"(ee"i),b)∈B})  (1 13 78)

*****REWRITE ↑ BY {97};

1 substitutions were made

105 DENUM(c)  (1 13 78 97)

*****TRY USING MONADIC 79 ↑ AUX1;

106 SET(ee"i)∧SET(c)  (1 13 78 97)

107 (SET(ee"i)∧SET(c))∧∃c1 d.(((ee"i)=c1∧c=d)∧(c1⊂G∧(¬(c1=λ)∧d=IF DEN%
UM({b|b∈c1∧pair(p"c1,b)∈R}) THEN {b|b∈c1∧pair(p"c1,b)∈R} ELSE {b|b∈c1%
∧pair(p"c1,b)∈B})))  (1 13 78 97)

108 IF DENUM({b|b∈(ee"i)∧pair(p"(ee"i),b)∈R}) THEN {b|b∈(ee"i)∧pair(p%
"(ee"i),b)∈R} ELSE {b|b∈(ee"i)∧pair(p"(ee"i),b)∈B}=c⊃((SET(ee"i)∧SET(%
c))∧∃c1 d.(((ee"i)=c1∧c=d)∧(c1⊂G∧(¬(c1=λ)∧d=IF DENUM({b|b∈c1∧pair(p"c1,b)%
∈R}) THEN {b|b∈c1∧pair(p"c1,b)∈R} ELSE {b|b∈c1∧pair(p"c1,b)∈B})))))  (1 13 78)

109 IF DENUM({b|b∈(ee"i)∧pair(p"(ee"i),b)∈R}) THEN {b|b∈(ee"i)∧pair(p%
"(ee"i),b)∈R} ELSE {b|b∈(ee"i)∧pair(p"(ee"i),b)∈B}=c≡((SET(ee"i)∧SET(%
c))∧∃c1 d.(((ee"i)=c1∧c=d)∧(c1⊂G∧(¬(c1=λ)∧d=IF DENUM({b|b∈c1∧pair(p"c1,b)%
∈R}) THEN {b|b∈c1∧pair(p"c1,b)∈R} ELSE {b|b∈c1∧pair(p"c1,b)∈B})))))  (1 13 78)

110 ∀c.(IF DENUM({b|b∈(ee"i)∧pair(p"(ee"i),b)∈R}) THEN {b|b∈(ee"i)∧pa%
ir(p"(ee"i),b)∈R} ELSE {b|b∈(ee"i)∧pair(p"(ee"i),b)∈B}=c≡((SET(ee"i)∧%
SET(c))∧∃c1 d.(((ee"i)=c1∧c=d)∧(c1⊂G∧(¬(c1=λ)∧d=IF DENUM({b|b∈c1∧pair%
(p"c1,b)∈R}) THEN {b|b∈c1∧pair(p"c1,b)∈R} ELSE {b|b∈c1∧pair(p"c1,b)∈B%
})))))  (1 13 78)

111 ∀c.(IF DENUM({b|b∈(ee"i)∧pair(p"(ee"i),b)∈R}) THEN {b|b∈(ee"i)∧pa%
ir(p"(ee"i),b)∈R} ELSE {b|b∈(ee"i)∧pair(p"(ee"i),b)∈B}=c≡opair(ee"i,c%
)∈{b|∃c d.(b=opair(c,d)∧(c⊂G∧(¬(c=λ)∧d=IF DENUM({b|b∈c∧pair(p"c,b)∈R}%
) THEN {b|b∈c∧pair(p"c,b)∈R} ELSE {b|b∈c∧pair(p"c,b)∈B})))})≡∀c.(IF D%
ENUM({b|b∈(ee"i)∧pair(p"(ee"i),b)∈R}) THEN {b|b∈(ee"i)∧pair(p"(ee"i),%
b)∈R} ELSE {b|b∈(ee"i)∧pair(p"(ee"i),b)∈B}=c≡((SET(ee"i)∧SET(c))∧∃c1 %
d.(((ee"i)=c1∧c=d)∧(c1⊂G∧(¬(c1=λ)∧d=IF DENUM({b|b∈c1∧pair(p"c1,b)∈R})%
 THEN {b|b∈c1∧pair(p"c1,b)∈R} ELSE {b|b∈c1∧pair(p"c1,b)∈B})))))

112 ∀c.(IF DENUM({b|b∈(ee"i)∧pair(p"(ee"i),b)∈R}) THEN {b|b∈(ee"i)∧pa%
ir(p"(ee"i),b)∈R} ELSE {b|b∈(ee"i)∧pair(p"(ee"i),b)∈B}=c≡opair(ee"i,c%
)∈{b|∃c d.(b=opair(c,d)∧(c⊂G∧(¬(c=λ)∧d=IF DENUM({b|b∈c∧pair(p"c,b)∈R}%
) THEN {b|b∈c∧pair(p"c,b)∈R} ELSE {b|b∈c∧pair(p"c,b)∈B})))})  (1 13 78)

113 ∃d.∀c.(d=c≡opair(ee"i,c)∈{b|∃c d.(b=opair(c,d)∧(c⊂G∧(¬(c=λ)∧d=IF %
DENUM({b|b∈c∧pair(p"c,b)∈R}) THEN {b|b∈c∧pair(p"c,b)∈R} ELSE {b|b∈c∧p%
air(p"c,b)∈B})))})  (1 13 78)

RESOLVE ∃d.∀c.(d=c≡opair(ee"i,c)∈{b|∃c d.(b=opair(c,d)∧(c⊂G∧(¬(c=λ)∧d%
```

```
=IF DENUM({b|b∈c∧pair(p"c,b)∈R}) THEN {b|b∈c∧pair(p"c,b)∈R} ELSE {b|b%
∈c∧pair(p"c,b)∈B}))))}⊃opair(ee"i,{b|∃c d.(b=opair(c,d)∧(c⊂G∧(¬(c=λ)∧%
d=IF DENUM({b|b∈c∧pair(p"c,b)∈R}) THEN {b|b∈c∧pair(p"c,b)∈R} ELSE {b|%
b∈c∧pair(p"c,b)∈B}))))}"(ee"i))∈{b|∃c d.(b=opair(c,d)∧(c⊂G∧(¬(c=λ)∧d=I%
F DENUM({b|b∈c∧pair(p"c,b)∈R}) THEN {b|b∈c∧pair(p"c,b)∈R} ELSE {b|b∈c%
∧pair(p"c,b)∈B})))) , ∃d.∀c.(d=c≡opair(ee"i,c)∈{b|∃c d.(b=opair(c,d)∧%
(c⊂G∧(¬(c=λ)∧d=IF DENUM({b|b∈c∧pair(p"c,b)∈R}) THEN {b|b∈c∧pair(p"c,b%
)∈R} ELSE {b|b∈c∧pair(p"c,b)∈B}))))}) →→ opair(ee"i,{b|∃c d.(b=opair(c%
,d)∧(c⊂G∧(¬(c=λ)∧d=IF DENUM({b|b∈c∧pair(p"c,b)∈R}) THEN {b|b∈c∧pair(p%
"c,b)∈R} ELSE {b|b∈c∧pair(p"c,b)∈B})))}"(ee"i))∈{b|∃c d.(b=opair(c,d)%
∧(c⊂G∧(¬(c=λ)∧d=IF DENUM({b|b∈c∧pair(p"c,b)∈R}) THEN {b|b∈c∧pair(p"c,%
b)∈R} ELSE {b|b∈c∧pair(p"c,b)∈B})))}

114 opair(ee"i,{b|∃c d.(b=opair(c,d)∧(c⊂G∧(¬(c=λ)∧d=IF DENUM({b|b∈c∧p%
air(p"c,b)∈R}) THEN {b|b∈c∧pair(p"c,b)∈R} ELSE {b|b∈c∧pair(p"c,b)∈B})%
)}"(ee"i))∈{b|∃c d.(b=opair(c,d)∧(c⊂G∧(¬(c=λ)∧d=IF DENUM({b|b∈c∧pair%
(p"c,b)∈R}) THEN {b|b∈c∧pair(p"c,b)∈R} ELSE {b|b∈c∧pair(p"c,b)∈B})))}  (1 13 78)


*****∀E 112 ↑:#1#2;

115 IF DENUM({b|b∈(ee"i)∧pair(p"(ee"i),b)∈R}) THEN {b|b∈(ee"i)∧pair(p%
"(ee"i),b)∈R} ELSE {b|b∈(ee"i)∧pair(p"(ee"i),b)∈B}≈({b|∃c d.(b=opair(%
c,d)∧(c⊂G∧(¬(c=λ)∧d=IF DENUM({b|b∈c∧pair(p"c,b)∈R}) THEN {b|b∈c∧pair(%
p"c,b)∈R} ELSE {b|b∈c∧pair(p"c,b)∈B})))}"(ee"i))≡opair(ee"i,{b|∃c d.(%
b=opair(c,d)∧(c⊂G∧(¬(c=λ)∧d=IF DENUM({b|b∈c∧pair(p"c,b)∈R}) THEN {b|b%
∈c∧pair(p"c,b)∈R} ELSE {b|b∈c∧pair(p"c,b)∈B})))}"(ee"i))∈{b|∃c d.(b=o%
pair(c,d)∧(c⊂G∧(¬(c=λ)∧d=IF DENUM({b|b∈c∧pair(p"c,b)∈R}) THEN {b|b∈c∧%
pair(p"c,b)∈R} ELSE {b|b∈c∧pair(p"c,b)∈B})))}  (1 13 78)

*****∀E EEDEF i;

116 (ee"SUC(i))≈({b|∃c d.(b=opair(c,d)∧(c⊂G∧(¬(c=λ)∧d=IF DENUM({b|b∈c%
∧pair(p"c,b)∈R}) THEN {b|b∈c∧pair(p"c,b)∈R} ELSE {b|b∈c∧pair(p"c,b)∈B%
})))}"(ee"i))  (17)

*****REWRITE ↑↑ BY {↑↑↑}uLOGICTREE;

2 substitutions were made

117 IF DENUM({b|b∈(ee"i)∧pair(p"(ee"i),b)∈R}) THEN {b|b∈(ee"i)∧pair(p%
"(ee"i),b)∈R} ELSE {b|b∈(ee"i)∧pair(p"(ee"i),b)∈B}≈({b|∃c d.(b=opair(%
c,d)∧(c⊂G∧(¬(c=λ)∧d=IF DENUM({b|b∈c∧pair(p"c,b)∈R}) THEN {b|b∈c∧pair(%
p"c,b)∈R} ELSE {b|b∈c∧pair(p"c,b)∈B})))}"(ee"i))  (1 13 78)

*****SUBSTR ↑ IN ↑↑;

118 (ee"SUC(i))≈IF DENUM({b|b∈(ee"i)∧pair(p"(ee"i),b)∈R}) THEN {b|b∈(%
ee"i)∧pair(p"(ee"i),b)∈R} ELSE {b|b∈(ee"i)∧pair(p"(ee"i),b)∈B}  (1 13 17 78)

*****SUBSTR ↑ IN 104;

119 DENUM(ee"SUC(i))  (1 13 17 78)
```

```
*****QED *8*2*1*1*2;

*****TRY USING REWRITE BY {↑↑,SUBSET}uARGIFTREE;

Goal *8*2*1*1*1*1: ∀c.(((DENUM({b|b∈(ee"i)∧pair(p"(ee"i),b)∈R})⊃(SET%
(c)∧(c∈(ee"i)∧pair(p"(ee"i),c)∈R)))∧(¬DENUM({b|b∈(ee"i)∧pair(p"(ee"i)%
,b)∈R})⊃(SET(c)∧(c∈(ee"i)∧pair(p"(ee"i),c)∈B))))⊃c∈G)

*****TRY USING ELIMINATION;

Goal *8*2*1*1*1*1*1: ((DENUM({b|b∈(ee"i)∧pair(p"(ee"i),b)∈R})⊃(SET(c%
)∧(c∈(ee"i)∧pair(p"(ee"i),c)∈R)))∧(¬DENUM({b|b∈(ee"i)∧pair(p"(ee"i),b%
)∈R})⊃(SET(c)∧(c∈(ee"i)∧pair(p"(ee"i),c)∈B))))⊃c∈G
Goal *8*2*1*1*1*1*1*1: c∈G

*****PREPARE;


120 (DENUM({b|b∈(ee"i)∧pair(p"(ee"i),b)∈R})⊃(SET(c)∧(c∈(ee"i)∧pair(p"%
(ee"i),c)∈R)))∧(¬DENUM({b|b∈(ee"i)∧pair(p"(ee"i),b)∈R})⊃(SET(c)∧(c∈(e%
e"i)∧pair(p"(ee"i),c)∈B))) (120)

121 ¬DENUM({b|b∈(ee"i)∧pair(p"(ee"i),b)∈R})⊃(SET(c)∧(c∈(ee"i)∧pair(p"%
(ee"i),c)∈B)) (120)

122 DENUM({b|b∈(ee"i)∧pair(p"(ee"i),b)∈R})⊃(SET(c)∧(c∈(ee"i)∧pair(p"(%
ee"i),c)∈R)) (120)

*****VE EDGERB pair(p"(ee"i),c);

123 pair(p"(ee"i),c)∈EDGESET(G)≡(pair(p"(ee"i),c)∈R∨pair(p"(ee"i),c)∈B) (1)

*****REWRITE ↑ BY {EDGESET AUX12};

1 substitutions were made

124 ((p"(ee"i))∈G∧(c∈G∧¬((p"(ee"i))=c)))≡(pair(p"(ee"i),c)∈R∨pair(p"(%
ee"i),c)∈B) (1)

*****TRY USING TAUT 120 ↑↑;;

125 c∈G (1 13 17 78 120)

126 ((DENUM({b|b∈(ee"i)∧pair(p"(ee"i),b)∈R})⊃(SET(c)∧(c∈(ee"i)∧pair(p%
"(ee"i),c)∈R)))∧(¬DENUM({b|b∈(ee"i)∧pair(p"(ee"i),b)∈R})⊃(SET(c)∧(c∈(%
ee"i)∧pair(p"(ee"i),c)∈B))))⊃c∈G (1 13 17 78)

127 ∀c.(((DENUM({b|b∈(ee"i)∧pair(p"(ee"i),b)∈R})⊃(SET(c)∧(c∈(ee"i)∧pa%
ir(p"(ee"i),c)∈R)))∧(¬DENUM({b|b∈(ee"i)∧pair(p"(ee"i),b)∈R})⊃(SET(c)∧%
(c∈(ee"i)∧pair(p"(ee"i),c)∈B))))⊃c∈G) (1 13 17 78)

128 (ee"SUC(i))⊂G≡∀c.(((DENUM({b|b∈(ee"i)∧pair(p"(ee"i),b)∈R})⊃(SET(c%
)∧(c∈(ee"i)∧pair(p"(ee"i),c)∈R)))∧(¬DENUM({b|b∈(ee"i)∧pair(p"(ee"i),b%
)∈R})⊃(SET(c)∧(c∈(ee"i)∧pair(p"(ee"i),c)∈B))))⊃c∈G) (1 13 17 78)
```

```
129 (ee"SUC(i))⊂G  (1 13 17 78)

130 (ee"SUC(i))⊂G∧DENUM(ee"SUC(i))  (1 13 17 78)

131 ((ee"i)⊂G∧DENUM(ee"i))⊃((ee"SUC(i))⊂G∧DENUM(ee"SUC(i)))  (1 13 17)

132 ∀i.(((ee"i)⊂G∧DENUM(ee"i))⊃((ee"SUC(i))⊂G∧DENUM(ee"SUC(i))))  (1 13 17)

133 ∀i.((ee"i)⊂G∧DENUM(ee"i))  (1 13 17)


*****GOAL ∀i.(ee"SUC(i)⊂(ee"i)\singl(p"(ee"i)));

Goal #10:  ∀i.(ee"SUC(i))⊂((ee"i) \ singl(p"(ee"i)))

*****DED 78 118;

134 ((ee"i)⊂G∧DENUM(ee"i))⊃(ee"SUC(i))=IF DENUM({b|b∈(ee"i)∧pair(p"(e%
e"i),b)∈R}) THEN {b|b∈(ee"i)∧pair(p"(ee"i),b)∈R} ELSE {b|b∈(ee"i)∧pai%
r(p"(ee"i),b)∈B}  (1 13 17)

*****REWRITE ↑ BY {↑↑}uLOGICTREE;

2 substitutions were made

135 (ee"SUC(i))=IF DENUM({b|b∈(ee"i)∧pair(p"(ee"i),b)∈R}) THEN {b|b∈(%
ee"i)∧pair(p"(ee"i),b)∈R} ELSE {b|b∈(ee"i)∧pair(p"(ee"i),b)∈B}  (1 13 17)

*****LABEL SUCI ;

*****∀I ↑ i;

136 ∀i.(ee"SUC(i))=IF DENUM({b|b∈(ee"i)∧pair(p"(ee"i),b)∈R}) THEN {b|%
b∈(ee"i)∧pair(p"(ee"i),b)∈R} ELSE {b|b∈(ee"i)∧pair(p"(ee"i),b)∈B}  (1 13 17)

*****∀E AUX11 p"(ee"i) ee"i;

137 SET(p"(ee"i))⊃∀c.(c∈((ee"i) \ singl(p"(ee"i)))≡(c∈(ee"i)∧¬(c=(p"(ee"i)))))

*****MONADIC ↑:≡1 133 AUX3 AUX4 103;

138 SET(p"(ee"i))  (1 13 17)

*****TAUT ↑↑:≡2 ↑↑;

139 ∀c.(c∈((ee"i) \ singl(p"(ee"i)))≡(c∈(ee"i)∧¬(c=(p"(ee"i)))))  (1 13 17)

*****TRY USING REWRITE BY {SUCI SUBSET ↑}uARGIFTREE;

Goal #10≡1:  ∀i c.(((DENUM({b|b∈(ee"i)∧pair(p"(ee"i),b)∈R})⊃(SET(c)∧(%
c∈(ee"i)∧pair(p"(ee"i),c)∈R)))∧(¬DENUM({b|b∈(ee"i)∧pair(p"(ee"i),b)∈R%
})⊃(SET(c)∧(c∈(ee"i)∧pair(p"(ee"i),c)∈B))))⊃(c∈(ee"i)∧¬(c=(p"(ee"i)))))
```

```
*****TRY USING ∀I;

Goal *10*1*1: ((DENUM({b|b∈(ee"i)∧pair(p"(ee"i),b)∈R})⊃(SET(c)∧(c∈(e%
e"i)∧pair(p"(ee"i),c)∈R)))∧(¬DENUM({b|b∈(ee"i)∧pair(p"(ee"i),b)∈R})⊃(%
SET(c)∧(c∈(ee"i)∧pair(p"(ee"i),c)∈B))))⊃(c∈(ee"i)∧¬(c=(p"(ee"i))))

*****TRY USING TAUTEQ 124;

140 ((DENUM({b|b∈(ee"i)∧pair(p"(ee"i),b)∈R})⊃(SET(c)∧(c∈(ee"i)∧pair(p%
"(ee"i),c)∈R)))∧(¬DENUM({b|b∈(ee"i)∧pair(p"(ee"i),b)∈R})⊃(SET(c)∧(c∈(%
ee"i)∧pair(p"(ee"i),c)∈B))))⊃(c∈(ee"i)∧¬(c=(p"(ee"i)))))  (1)

141 ∀i c.(((DENUM({b|b∈(ee"i)∧pair(p"(ee"i),b)∈R})⊃(SET(c)∧(c∈(ee"i)∧%
pair(p"(ee"i),c)∈R)))∧(¬DENUM({b|b∈(ee"i)∧pair(p"(ee"i),b)∈R})⊃(SET(c%
)∧(c∈(ee"i)∧pair(p"(ee"i),c)∈B))))⊃(c∈(ee"i)∧¬(c=(p"(ee"i)))))  (1)

142 ∀i.(ee"SUC(i))⊂((ee"i) \ singl(p"(ee"i)))≡∀i c.(((DENUM({b|b∈(ee"%
i)∧pair(p"(ee"i),b)∈R})⊃(SET(c)∧(c∈(ee"i)∧pair(p"(ee"i),c)∈R)))∧(¬DEN%
UM({b|b∈(ee"i)∧pair(p"(ee"i),b)∈R})⊃(SET(c)∧(c∈(ee"i)∧pair(p"(ee"i),c%
)∈B))))⊃(c∈(ee"i)∧¬(c=(p"(ee"i)))))  (1 13 17)

143 ∀i.(ee"SUC(i))⊂((ee"i) \ singl(p"(ee"i)))  (1 13 17)

*****GOAL ∀j i.(j LT i⊃ee"i⊂ee"SUC(j));

Goal *11:  ∀j i.(j LT i⊃(ee"i)⊂(ee"SUC(j)))

*****TRY USING ∀I j;

Goal *11*1:  ∀i.(j LT i⊃(ee"i)⊂(ee"SUC(j)))

*****TRY USING INDUCTION;

Goal *11*1*1:  j LT λ⊃(ee"λ)⊂(ee"SUC(j))
Goal *11*1*2:  ∀i.((j LT i⊃(ee"i)⊂(ee"SUC(j)))⊃(j LT SUC(i)⊃(ee"SUC(i%
))⊂(ee"SUC(j))))

*****TRY 1 USING REWRITE BY {LESS4};

144 j LT λ⊃(ee"λ)⊂(ee"SUC(j))


*****TRY  USING REWRITE BY {LESS7};

Goal *11*1*2*1:  ∀i.((j LT i⊃(ee"i)⊂(ee"SUC(j)))⊃((j=ivj LT i)⊃(ee"SU%
C(i))⊂(ee"SUC(j))))

*****TRY USING ELIMINATION;

Goal *11*1*2*1*1:  (j LT i⊃(ee"i)⊂(ee"SUC(j)))⊃((j=ivj LT i)⊃(ee"SUC(%
i))⊂(ee"SUC(j)))
Goal *11*1*2*1*1*1:  (j=ivj LT i)⊃(ee"SUC(i))⊂(ee"SUC(j))
145 j LT i⊃(ee"i)⊂(ee"SUC(j))  (145)
```

```
Goal #11#1#2#1#1#1#1: (ee"SUC(i))⊂(ee"SUC(j))
146 j=ivj LT i  (146)

Goal #11#1#2#1#1#1#1#1: j=i⊃(ee"SUC(i))⊂(ee"SUC(j))
Goal #11#1#2#1#1#1#1#2: j LT i⊃(ee"SUC(i))⊂(ee"SUC(j))
Goal #11#1#2#1#1#1#1#1#1: (ee"SUC(i))⊂(ee"SUC(j))
Goal #11#1#2#1#1#1#1#2#1: (ee"SUC(i))⊂(ee"SUC(j))

*****PREPARE;

147 j LT i  (147)

RESOLVE j LT i⊃(ee"i)⊂(ee"SUC(j)) , j LT i ↛ (ee"i)⊂(ee"SUC(j))

148 (ee"i)⊂(ee"SUC(j))  (145 147)

*****VE 143 i;

149 (ee"SUC(i))⊂((ee"i) \ singl(p"(ee"i)))  (1 13 17)

*****VE AUX29 ee"i singl(p"(ee"i));

150 ((ee"i) \ singl(p"(ee"i)))⊂(ee"i)

*****VE AUX23 149:#1 149:#2 150:#2;

151 ((ee"SUC(i))⊂((ee"i) \ singl(p"(ee"i)))∧((ee"i) \ singl(p"(ee"i))%
)⊂(ee"i))⊃(ee"SUC(i))⊂(ee"i)

*****VE AUX23 ↑:#2#1 ↑:#2#2 148:#2;

152 ((ee"SUC(i))⊂(ee"i)∧(ee"i)⊂(ee"SUC(j)))⊃(ee"SUC(i))⊂(ee"SUC(j))

*****TRY USING TAUT 148;;

153 (ee"SUC(i))⊂(ee"SUC(j))  (1 13 17 145 147)

154 j LT i⊃(ee"SUC(i))⊂(ee"SUC(j))  (1 13 17 145)


*****TRY USING REWRITE BY {SUBSET};

155 j=i  (155)

156 (ee"SUC(i))⊂(ee"SUC(j))  (155)

157 j=i⊃(ee"SUC(i))⊂(ee"SUC(j))

158 (ee"SUC(i))⊂(ee"SUC(j))  (1 13 17 145 146)

159 (j=ivj LT i)⊃(ee"SUC(i))⊂(ee"SUC(j))  (1 13 17 145)

160 (j LT i⊃(ee"i)⊂(ee"SUC(j)))⊃((j=ivj LT i)⊃(ee"SUC(i))⊂(ee"SUC(j)))  (1 13 17)
```

```
161 ∀i.((j LT i⊃(ee"i)⊂(ee"SUC(j)))⊃((j=ivj LT i)⊃(ee"SUC(i))⊂(ee"SUC(j)))) (1 13 17)

162 ∀i.((j LT i⊃(ee"i)⊂(ee"SUC(j)))⊃(j LT SUC(i)⊃(ee"SUC(i))⊂(ee"SUC(j))))%
≡∀i.((j LT i⊃(ee"i)⊂(ee"SUC(j)))⊃((j=ivj LT i)⊃(ee"SUC(i))⊂(ee"SUC(j))))

163 ∀i.((j LT i⊃(ee"i)⊂(ee"SUC(j)))⊃(j LT SUC(i)⊃(ee"SUC(i))⊂(ee"SUC(j)))) (1 13 17)

164 ∀i.(j LT i⊃(ee"i)⊂(ee"SUC(j))) (1 13 17)

165 ∀j i.(j LT i⊃(ee"i)⊂(ee"SUC(j))) (1 13 17)

*****GOAL FNC(CONV(48:#1))∧DOM(48:#1)=omega∧RNG(48:#1)⊂G;

Goal #12: FNC(CONV({b|∃k.b=opair(k,p"(ee"k))}))∧(DOM({b|∃k.b=opair(k%
,p"(ee"k))})=omega∧RNG({b|∃k.b=opair(k,p"(ee"k))})⊂G)

*****∀E DOM 48:#1;

166 FNC({b|∃k.b=opair(k,p"(ee"k))})⊃DOM({b|∃k.b=opair(k,p"(ee"k))})={%
c|∃a.opair(c,a)∈{b|∃k.b=opair(k,p"(ee"k))}}

*****∀E RNG 48:#1;

167 FNC({b|∃k.b=opair(k,p"(ee"k))})⊃RNG({b|∃k.b=opair(k,p"(ee"k))})={%
c|∃a.opair(a,c)∈{b|∃k.b=opair(k,p"(ee"k))}}

*****∀E CONV 48:#1;

168 REL({b|∃k.b=opair(k,p"(ee"k))})⊃CONV({b|∃k.b=opair(k,p"(ee"k))})=%
{c|∃a b.(c=opair(a,b)∧opair(b,a)∈{b|∃k.b=opair(k,p"(ee"k))})}

*****REWRITE 48 BY {FNC};

1 substitutions were made

169 REL({b|∃k.b=opair(k,p"(ee"k))})∧∀b c d.((opair(b,c)∈{b|∃k.b=opair%
(k,p"(ee"k))}∧opair(b,d)∈{b|∃k.b=opair(k,p"(ee"k))})⊃c=d)

*****TAUT ↑↑↑↑:#2 ↑↑↑↑ 48;

170 DOM({b|∃k.b=opair(k,p"(ee"k))})={c|∃a.opair(c,a)∈{b|∃k.b=opair(k,p"(ee"k))}}

*****TAUT ↑↑↑↑:#2 ↑↑↑↑ 48;

171 RNG({b|∃k.b=opair(k,p"(ee"k))})={c|∃a.opair(a,c)∈{b|∃k.b=opair(k,p"(ee"k))}}

*****TAUT ↑↑↑↑:#2 ↑↑↑↑,↑↑↑;

172 CONV({b|∃k.b=opair(k,p"(ee"k))})={c|∃a b.(c=opair(a,b)∧opair(b,a)%
∈{b|∃k.b=opair(k,p"(ee"k))})}

*****REWRITE c=omega BY {KEXT};
```

1 substitutions were made

173 c=omega≡∀c1.(c1∈c≡c1∈omega)

*****∀I ↑ c;

174 ∀c.(c=omega≡∀c1.(c1∈c≡c1∈omega))

*****TRY USING REWRITE BY {FNC REL SUBSET AUX5 AUX27 omega ↑,↑↑↑,↑↑↑↑,↑↑↑↑↑};

Goal *12*1: (∀d.((SET(d)∧∃a b.(d=opair(a,b)∧((SET(b)∧SET(a))∧∃k.(b=k%
∧a=(p"(ee"k))))))⊃∃b c.d=opair(b,c))∧∀b1 c d.((((SET(b1)∧SET(c))∧∃a b%
.((b1=a∧c=b)∧((SET(b)∧SET(a))∧∃k.(b=k∧a=(p"(ee"k))))))∧((SET(b1)∧SET(%
d))∧∃a b.((b1=a∧d=b)∧((SET(b)∧SET(a))∧∃k.(b=k∧a=(p"(ee"k)))))))⊃c=d))%
∧(∀c1.((SET(c1)∧∃a.((SET(c1)∧SET(a))∧∃k.(c1=k∧a=(p"(ee"k)))))≡(SET(c1)%
∧NATNUM(c1)))∧∀c.((SET(c)∧∃a.((SET(a)∧SET(c))∧∃k.(a=k∧c=(p"(ee"k)))))⊃c∈G))

*****TRY USING ELIMINATION DEPTH 3;

Goal *12*1*1: ∀d.((SET(d)∧∃a b.(d=opair(a,b)∧((SET(b)∧SET(a))∧∃k.(b=%
k∧a=(p"(ee"k))))))⊃∃b c.d=opair(b,c))∧∀b1 c d.((((SET(b1)∧SET(c))∧∃a %
b.((b1=a∧c=b)∧((SET(b)∧SET(a))∧∃k.(b=k∧a=(p"(ee"k))))))∧((SET(b1)∧SET%
(d))∧∃a b.((b1=a∧d=b)∧((SET(b)∧SET(a))∧∃k.(b=k∧a=(p"(ee"k)))))))⊃c=d)
Goal *12*1*2: ∀c1.((SET(c1)∧∃a.((SET(c1)∧SET(a))∧∃k.(c1=k∧a=(p"(ee"k%
)))))≡(SET(c1)∧NATNUM(c1)))∧∀c.((SET(c)∧∃a.((SET(a)∧SET(c))∧∃k.(a=k∧c%
=(p"(ee"k)))))⊃c∈G)
Goal *12*1*1*1: ∀d.((SET(d)∧∃a b.(d=opair(a,b)∧((SET(b)∧SET(a))∧∃k.(%
b=k∧a=(p"(ee"k))))))⊃∃b c.d=opair(b,c))
Goal *12*1*1*2: ∀b1 c d.((((SET(b1)∧SET(c))∧∃a b.((b1=a∧c=b)∧((SET(b%
)∧SET(a))∧∃k.(b=k∧a=(p"(ee"k))))))∧((SET(b1)∧SET(d))∧∃a b.((b1=a∧d=b)%
∧((SET(b)∧SET(a))∧∃k.(b=k∧a=(p"(ee"k)))))))⊃c=d)
Goal *12*1*1*1*1: (SET(d)∧∃a b.(d=opair(a,b)∧((SET(b)∧SET(a))∧∃k.(b=%
k∧a=(p"(ee"k))))))⊃∃b c.d=opair(b,c)
Goal *12*1*1*2*1: (((SET(b1)∧SET(c))∧∃a b.((b1=a∧c=b)∧((SET(b)∧SET(a%
))∧∃k.(b=k∧a=(p"(ee"k))))))∧((SET(b1)∧SET(d))∧∃a b.((b1=a∧d=b)∧((SET(%
b)∧SET(a))∧∃k.(b=k∧a=(p"(ee"k)))))))⊃c=d
Goal *12*1*2*1: ∀c1.((SET(c1)∧∃a.((SET(c1)∧SET(a))∧∃k.(c1=k∧a=(p"(ee%
"k)))))≡(SET(c1)∧NATNUM(c1)))
Goal *12*1*2*2: ∀c.((SET(c)∧∃a.((SET(a)∧SET(c))∧∃k.(a=k∧c=(p"(ee"k)))))⊃c∈G)
Goal *12*1*2*1*1: (SET(c1)∧∃a.((SET(c1)∧SET(a))∧∃k.(c1=k∧a=(p"(ee"k)%
)))))≡(SET(c1)∧NATNUM(c1))
Goal *12*1*2*2*1: (SET(c)∧∃a.((SET(a)∧SET(c))∧∃k.(a=k∧c=(p"(ee"k)))))⊃c∈G

*****REWRITE 133 BY {SUBSET};

1 substitutions were made

175 ∀i.(∀c.(c∈(ee"i)⊃c∈G)∧DENUM(ee"i)) (1 13 17)

*****REWRITE 103 BY {SUBSET};

1 substitutions were made

176 (∀c.(c∈(ee"i)⊃c∈G)∧¬((ee"i)=λ))⊃(p"(ee"i))∈(ee"i) (13)

```
*****∀E AUX3 ee"i;

177 DENUM(ee"i)⊃¬((ee"i)=λ)

*****∀E ↑↑↑ i;

178 ∀c.(c∈(ee"i)⊃c∈G)∧DENUM(ee"i)  (1 13 17)

*****MONADIC p"(ee"i)∈G ↑↑↑;;

179 (p"(ee"i))∈G  (1 13 17)

*****∀I ↑ i;

180 ∀i.(p"(ee"i))∈G  (1 13 17)

*****TRY USING MONADIC ↑;

181 (SET(c)∧∃a.((SET(a)∧SET(c))∧∃k.(a=k∧c=(p"(ee"k)))))⊃c∈G  (1 13 17)

182 ∀c.((SET(c)∧∃a.((SET(a)∧SET(c))∧∃k.(a=k∧c=(p"(ee"k)))))⊃c∈G)  (1 13 17)


*****∀E SET p"(ee"i);

183 SET(p"(ee"i))≡∃b.(p"(ee"i))∈b

*****MONADIC ↑:#1 ↑↑↑↑↑,↑;

184 SET(p"(ee"i))  (1 13 17)

*****LABEL SETPEEI;

*****∀I ↑ i;

185 ∀i.SET(p"(ee"i))  (1 13 17)

*****TRY USING ELIMINATION;

Goal #12#1#2#1#1#1: (SET(c1)∧∃a.((SET(c1)∧SET(a))∧∃k.(c1=k∧a=(p"(ee"%
k)))))⊃(SET(c1)∧NATNUM(c1))
Goal #12#1#2#1#1#2: (SET(c1)∧NATNUM(c1))⊃(SET(c1)∧∃a.((SET(c1)∧SET(a%
))∧∃k.(c1=k∧a=(p"(ee"k)))))
Goal #12#1#2#1#1#1#1: SET(c1)∧NATNUM(c1)
186 SET(c1)∧∃a.((SET(c1)∧SET(a))∧∃k.(c1=k∧a=(p"(ee"k))))  (186)

187 ∃a.((SET(c1)∧SET(a))∧∃k.(c1=k∧a=(p"(ee"k))))  (186)

188 SET(c1)  (186)

Goal #12#1#2#1#1#1#1#1: SET(c1)
Goal #12#1#2#1#1#1#1#2: NATNUM(c1)
Goal #12#1#2#1#1#2#1: SET(c1)∧∃a.((SET(c1)∧SET(a))∧∃k.(c1=k∧a=(p"(ee"k))))
```

```
189 SET(c1)∧NATNUM(c1) (189)

190 NATNUM(c1) (189)

191 SET(c1) (189)

Goal *12*1*2*1*1*2*1*1: SET(c1)
Goal *12*1*2*1*1*2*1*2: ∃a.((SET(c1)∧SET(a))∧∃k.(c1=k∧a=(p"(ee"k))))
Goal *12*1*2*1*1*2*1*2*1: (SET(c1)∧SET(a))∧∃k.(c1=k∧a=(p"(ee"k)))
Goal *12*1*2*1*1*2*1*2*1*1: SET(c1)∧SET(a)
Goal *12*1*2*1*1*2*1*2*1*2: ∃k.(c1=k∧a=(p"(ee"k)))
Goal *12*1*2*1*1*2*1*2*1*1*1: SET(c1)
Goal *12*1*2*1*1*2*1*2*1*1*2: SET(a)
Goal *12*1*2*1*1*2*1*2*1*2*1: c1=k∧a=(p"(ee"k))
Goal *12*1*2*1*1*2*1*2*1*2*1*1: c1=k
Goal *12*1*2*1*1*2*1*2*1*2*1*2: a=(p"(ee"k))

*****TAUTEQ c1=c1;

192 c1=c1

*****EG ↑ c1←k OCC 2;

193 NATNUM(c1)⊃∃k.c1=k

*****TAUT ↑:#2 ↑↑↑↑,↑;

194 ∃k.c1=k (189)

*****TRY 1 USING UNIFY ↑;

195 ∃k.c1=k (189)

196 c1=k (196)

*****TRY USING EQUNIFY;

197 ∃a.a=(p"(ee"k))

198 a=(p"(ee"k)) (198)

199 c1=k∧a=(p"(ee"k)) (196 198)

200 ∃k.(c1=k∧a=(p"(ee"k))) (196 198)

*****TRY USING MONADIC SETPEEI ↑↑↑;

201 SET(a) (1 13 17 198)

*****SIMPLIFY ∀i.NATNUM(i);
```

```
202 ∀i.NATNUM(i)

*****TRY *12*1*2*1*1*1*1*2 USING MONADIC 187 ↑;

203 NATNUM(c1)  (186)


*****TRY *12*1*1*1*1 USING MONADIC;

204 (SET(d)∧∃a b.(d=opair(a,b)∧((SET(b)∧SET(a))∧∃k.(b=k∧a=(p"(ee"k)))%
)))⊃∃b c.d=opair(b,c)

205 ∀d.((SET(d)∧∃a b.(d=opair(a,b)∧((SET(b)∧SET(a))∧∃k.(b=k∧a=(p"(ee"%
k))))))⊃∃b c.d=opair(b,c))


*****GOAL ∀i j.(j LT i⊃¬ee"j=ee"i) ASSUME 165;

Goal *13:  ∀i j.(j LT i⊃¬((ee"j)=(ee"i)))

*****TRY USING LOGIC;

The wff of this goal does not rewrite.  Sorry.
Goal *13*1:  j LT i⊃¬((ee"j)=(ee"i))
The wff of this goal does not rewrite.  Sorry.
Goal *13*1*1:  ¬((ee"j)=(ee"i))
206 j LT i  (206)

RESOLVE j LT i⊃(ee"i)⊂(ee"SUC(j)) , j LT i ↣ (ee"i)⊂(ee"SUC(j))

207 (ee"i)⊂(ee"SUC(j))  (1 13 17 206)

The wff of this goal does not rewrite.  Sorry.
We have a failqueue of length:  1
Starting a new 2-th pass on new queue of length: 1
The wff of this goal does not rewrite.  Sorry.
We have a failqueue of length:  1
Failure: can't prove anything on failqueue.
The tactic LOGIC can't be applied to goal
Goal *13:  ∀i j.(j LT i⊃¬((ee"j)=(ee"i)))
    FACTS:  165  ∀j i.(j LT i⊃(ee"i)⊂(ee"SUC(j)))
       Simpsets:  ( BY  LOGICTREE  COMPTREE)
       Reasons:  (∀I ((j i) j i) NIL)
       Number of sons:  1

*****∀E 143 j;

208 (ee"SUC(j))⊂((ee"j) \ singl(p"(ee"j)))  (1 13 17)

*****∀E AUX23 ↑↑:*1 ↑:*1 ↑:*2;

209 ((ee"i)⊂(ee"SUC(j))∧(ee"SUC(j))⊂((ee"j) \ singl(p"(ee"j))))⊃(ee"i%
)⊂((ee"j) \ singl(p"(ee"j)))
```

*****TAUT ↑:=2 ↑↑↑;;

210 (ee"i)⊂((ee"j) \ singl(p"(ee"j))) (1 13 17 206)

*****MONADIC p"(ee"i) ∈ ee"i 103 133 AUX3;

211 (p"(ee"i))∈(ee"i) (1 13 17)

*****LABEL PINEE;

*****∀I ↑ i;

212 ∀i.(p"(ee"i))∈(ee"i) (1 13 17)

*****∀E AUX11 p"(ee"j) ee"j;

213 SET(p"(ee"j))⊃∀c.(c∈((ee"j) \ singl(p"(ee"j)))≡(c∈(ee"j)∧¬(c=(p"(ee"j)))))

*****RESOLVE SETPEEI ↑;

RESOLVE SET(p"(ee"j))⊃∀c.(c∈((ee"j) \ singl(p"(ee"j)))≡(c∈(ee"j)∧¬(c=%
(p"(ee"j)))) , ∀i.SET(p"(ee"i)) →→ ∀c.(c∈((ee"j) \ singl(p"(ee"j)))≡%
(c∈(ee"j)∧¬(c=(p"(ee"j)))))

214 ∀c.(c∈((ee"j) \ singl(p"(ee"j)))≡(c∈(ee"j)∧¬(c=(p"(ee"j))))) (1 13 17)

*****REWRITE 210 BY {SUBSET};

1 substitutions were made

215 ∀c.(c∈(ee"i)⊃c∈((ee"j) \ singl(p"(ee"j)))) (1 13 17 206)

*****MONADIC ¬ p"(ee"j) ∈ ee"i ↑↑;;

216 ¬((p"(ee"j))∈(ee"i)) (1 13 17 206)

*****TRY USING REWRITE BY {KEXT PINEE ↑};

Goal *13*1*1*1: ¬∀c.(c∈(ee"j)≡c∈(ee"i))

*****TRY USING MONADIC PINEE ↑;

217 ¬∀c.(c∈(ee"j)≡c∈(ee"i)) (1 13 17 206)

218 ¬((ee"j)=(ee"i))≡¬∀c.(c∈(ee"j)≡c∈(ee"i))

219 ¬((ee"j)=(ee"i)) (1 13 17 206)

220 j LT i⊃¬((ee"j)=(ee"i)) (1 13 17)

221 ∀i j.(j LT i⊃¬((ee"j)=(ee"i))) (1 13 17)

```
*****MONADIC ∀i j.(¬i=j⊃¬ee"i=ee"j) LESS2 ↑;

222 ∀i j.(¬(i=j)⊃¬((ee"i)=(ee"j))) (1 13 17)

*****MONADIC ¬p"(ee"j)=p"(ee"i) PINEE 216;

223 ¬((p"(ee"j))=(p"(ee"i))) (1 13 17 206)

*****DED 206 ↑;

224 j LT i⊃¬((p"(ee"j))=(p"(ee"i))) (1 13 17)

*****∀I ↑ j i;

225 ∀j i.(j LT i⊃¬((p"(ee"j))=(p"(ee"i)))) (1 13 17)

*****∀E ↑ i j;

226 i LT j⊃¬((p"(ee"i))=(p"(ee"j))) (1 13 17)

*****MONADIC ↑:#2#1⊃i=j LESS2 ↑↑↑,↑;

227 (p"(ee"i))=(p"(ee"j))⊃i=j (1 13 17)

*****LABEL INJPEE;

*****∀I ↑ i j;

228 ∀i j.((p"(ee"i))=(p"(ee"j))⊃i=j) (1 13 17)

*****LABEL INJEE 222;

*****TRY #12#1#1#2#1 USING ⊃I;

Goal #12#1#1#2#1#1: c=d

*****PREPARE;

229 ((SET(b1)∧SET(c))∧∃a b.((b1=a∧c=b)∧((SET(b)∧SET(a))∧∃k.(b=k∧a=(p"%
(ee"k))))))∧((SET(b1)∧SET(d))∧∃a b.((b1=a∧d=b)∧((SET(b)∧SET(a))∧∃k.(b%
=k∧a=(p"(ee"k))))))  (229)

230 ∃a b.((b1=a∧d=b)∧((SET(b)∧SET(a))∧∃k.(b=k∧a=(p"(ee"k))))))  (229)

231 SET(d)  (229)

232 SET(b1)  (229)

233 ∃a b.((b1=a∧c=b)∧((SET(b)∧SET(a))∧∃k.(b=k∧a=(p"(ee"k)))))  (229)

234 SET(c)  (229)

235 SET(b1)  (229)
```

*****ES ↑↑↑↑↑↑ a b;

236 (b1=a∧d=b)∧((SET(b)∧SET(a))∧∃k.(b=k∧a=(p"(ee"k)))) (236)

*****ES ↑↑↑↑ a1 c1;

237 (b1=a1∧c=c1)∧((SET(c1)∧SET(a1))∧∃k.(c1=k∧a1=(p"(ee"k)))) (237)

*****TAUT ↑↑:*2*2 ↑↑;

238 ∃k.(b=k∧a=(p"(ee"k))) (236)

*****TAUT ↑↑:*2*2 ↑↑;

239 ∃k.(c1=k∧a1=(p"(ee"k))) (237)

*****ES ↑↑ i;

240 b=i∧a=(p"(ee"i)) (240)

*****ES ↑↑ j;

241 c1=j∧a1=(p"(ee"j)) (241)

*****∀E INJPEE i j;

242 (p"(ee"i))=(p"(ee"j))⊃i=j (1 13 17)

*****TRY USING TAUTEQ ↑,↑↑,↑↑↑,↑↑↑↑↑↑,↑↑↑↑↑↑↑;

243 c=d (1 13 17 229)

244 (((SET(b1)∧SET(c))∧∃a b.((b1=a∧c=b)∧((SET(b)∧SET(a))∧∃k.(b=k∧a=(p%
"(ee"k))))))∧((SET(b1)∧SET(d))∧∃a b.((b1=a∧d=b)∧((SET(b)∧SET(a))∧∃k.(%
b=k∧a=(p"(ee"k)))))))⊃c=d (1 13 17)

245 ∀b1 c d.((((SET(b1)∧SET(c))∧∃a b.((b1=a∧c=b)∧((SET(b)∧SET(a))∧∃k.%
(b=k∧a=(p"(ee"k))))))∧((SET(b1)∧SET(d))∧∃a b.((b1=a∧d=b)∧((SET(b)∧SET%
(a))∧∃k.(b=k∧a=(p"(ee"k)))))))⊃c=d) (1 13 17)

246 ∀d.((SET(d)∧∃a b.(d=opair(a,b)∧((SET(b)∧SET(a))∧∃k.(b=k∧a=(p"(ee"%
k))))))⊃∃b c.d=opair(b,c))∧∀b1 c d.((((SET(b1)∧SET(c))∧∃a b.((b1=a∧c=%
b)∧((SET(b)∧SET(a))∧∃k.(b=k∧a=(p"(ee"k))))))∧((SET(b1)∧SET(d))∧∃a b.(%
(b1=a∧d=b)∧((SET(b)∧SET(a))∧∃k.(b=k∧a=(p"(ee"k)))))))⊃c=d) (1 13 17)

*****TRY USING LOGIC;

247 ∀b1 c d.((((SET(b1)∧SET(c))∧∃a b.((b1=a∧c=b)∧((SET(b)∧SET(a))∧∃k.%
(b=k∧a=(p"(ee"k))))))∧((SET(b1)∧SET(d))∧∃a b.((b1=a∧d=b)∧((SET(b)∧SET%
(a))∧∃k.(b=k∧a=(p"(ee"k)))))))⊃c=d) (1 13 17)

248 ∀d.(¡SET(d)∧∃a b.(d=opair(a,b)∧((SET(b)∧SET(a))∧∃k.(b=k∧a=(p"(ee"%

k))))))⊃∃b c.d=opair(b,c)) (1 13 17)

249 SET(c1) (189)

250 SET(c1)∧SET(a) (1 13 17 189 198)

251 (SET(c1)∧SET(a))∧∃k.(c1=k∧a=(p"(ee"k))) (1 13 17 189 196 198)

252 ∃a.((SET(c1)∧SET(a))∧∃k.(c1=k∧a=(p"(ee"k)))) (1 13 17 189)

253 ∀b1 c d.((((SET(b1)∧SET(c))∧∃a b.((b1=a∧c=b)∧((SET(b)∧SET(a))∧∃k.%
(b=k∧a=(p"(ee"k))))))∧((SET(b1)∧SET(d))∧∃a b.((b1=a∧d=b)∧((SET(b)∧SET%
(a))∧∃k.(b=k∧a=(p"(ee"k)))))))⊃c=d) (1 13 17)

254 ∀d.((SET(d)∧∃a b.(d=opair(a,b)∧((SET(b)∧SET(a))∧∃k.(b=k∧a=(p"(ee"%
k))))))⊃∃b c.d=opair(b,c)) (1 13 17)

255 SET(c1) (189)

256 SET(c1)∧∃a.((SET(c1)∧SET(a))∧∃k.(c1=k∧a=(p"(ee"k)))) (1 13 17 189)

257 (SET(c1)∧NATNUM(c1))⊃(SET(c1)∧∃a.((SET(c1)∧SET(a))∧∃k.(c1=k∧a=(p"%
(ee"k))))) (1 13 17)

258 ∀b1 c d.((((SET(b1)∧SET(c))∧∃a b.((b1=a∧c=b)∧((SET(b)∧SET(a))∧∃k.%
(b=k∧a=(p"(ee"k))))))∧((SET(b1)∧SET(d))∧∃a b.((b1=a∧d=b)∧((SET(b)∧SET%
(a))∧∃k.(b=k∧a=(p"(ee"k)))))))⊃c=d) (1 13 17)

259 ∀d.((SET(d)∧∃a b.(d=opair(a,b)∧((SET(b)∧SET(a))∧∃k.(b=k∧a=(p"(ee"%
k))))))⊃∃b c.d=opair(b,c)) (1 13 17)

260 SET(c1) (186)

261 SET(c1)∧NATNUM(c1) (186)

262 (SET(c1)∧∃a.((SET(c1)∧SET(a))∧∃k.(c1=k∧a=(p"(ee"k)))))⊃(SET(c1)∧NATNUM(c1))

263 (SET(c1)∧∃a.((SET(c1)∧SET(a))∧∃k.(c1=k∧a=(p"(ee"k)))))≡(SET(c1)∧N%
ATNUM(c1)) (1 13 17)

264 ∀c1.((SET(c1)∧∃a.((SET(c1)∧SET(a))∧∃k.(c1=k∧a=(p"(ee"k)))))≡(SET(%
c1)∧NATNUM(c1))) (1 13 17)

265 ∀c1.((SET(c1)∧∃a.((SET(c1)∧SET(a))∧∃k.(c1=k∧a=(p"(ee"k)))))≡(SET(c1)%
∧NATNUM(c1)))∧∀c.((SET(c)∧∃a.((SET(a)∧SET(c))∧∃k.(a=k∧c=(p"(ee"k)))))⊃c∈G) (1 13 17)

266 (∀d.((SET(d)∧∃a b.(d=opair(a,b)∧((SET(b)∧SET(a))∧∃k.(b=k∧a=(p"(ee%
"k))))))⊃∃b c.d=opair(b,c))∧∀b1 c d.((((SET(b1)∧SET(c))∧∃a b.((b1=a∧c%
=b)∧((SET(b)∧SET(a))∧∃k.(b=k∧a=(p"(ee"k))))))∧((SET(b1)∧SET(d))∧∃a b.%
((b1=a∧d=b)∧((SET(b)∧SET(a))∧∃k.(b=k∧a=(p"(ee"k)))))))⊃c=d))∧(∀c1.((S%
ET(c1)∧∃a.((SET(c1)∧SET(a))∧∃k.(c1=k∧a=(p"(ee"k)))))≡(SET(c1)∧NATNUM(%
c1)))∧∀c.((SET(c)∧∃a.((SET(a)∧SET(c))∧∃k.(a=k∧c=(p"(ee"k)))))⊃c∈G)) (1 13 17)

267 (FNC(CONV({b|∃k.b=opair(k,p"(ee"k))}))∧(DOM({b|∃k.b=opair(k,p"(ee%

```
"k)))}=omega∧RNG({b|∃k.b=opair(k,p"(ee"k))})⊂G))≡((∀d.((SET(d)∧∃a b.(%
d=opair(a,b)∧((SET(b)∧SET(a))∧∃k.(b=k∧a=(p"(ee"k))))))⊃∃b c.d=opair(b%
,c))∧∀b1 c d.((((SET(b1)∧SET(c))∧∃a b.((b1=a∧c=b)∧((SET(b)∧SET(a))∧∃k%
.(b=k∧a=(p"(ee"k))))))∧((SET(b1)∧SET(d))∧∃a b.((b1=a∧d=b)∧((SET(b)∧SE%
T(a))∧∃k.(b=k∧a=(p"(ee"k)))))))⊃c=d))∧(∀c1.((SET(c1)∧∃a.((SET(c1)∧SET%
(a))∧∃k.(c1=k∧a=(p"(ee"k)))))≡(SET(c1)∧NATNUM(c1)))∧∀c.((SET(c)∧∃a.((%
SET(a)∧SET(c))∧∃k.(a=k∧c=(p"(ee"k)))))⊃c∈G)))

268 FNC(CONV({b|∃k.b=opair(k,p"(ee"k))}))∧(DOM({b|∃k.b=opair(k,p"(ee"%
k))})=omega∧RNG({b|∃k.b=opair(k,p"(ee"k))})⊂G)  (1 13 17)

LOGIC SUCCEEDED!

*****TRY *1*1*1*1 USING ∧I;

Goal *1*1*1*1*1: RNG({b|∃k.b=opair(k,p"(ee"k))}) | IF DENUM({k|DENUM(%
{b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}) THEN {k|DENUM({b|b∈(ee"k)∧pair(p"(%
ee"k),b)∈R})} ELSE {k|¬DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}}⊂G
Goal *1*1*1*1*2: DENUM(RNG({b|∃k.b=opair(k,p"(ee"k))}) | IF DENUM({k|%
DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}) THEN {k|DENUM({b|b∈(ee"k)∧pa%
ir(p"(ee"k),b)∈R})} ELSE {k|¬DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})})%
)∧(EDGESET(RNG({b|∃k.b=opair(k,p"(ee"k))}) | IF DENUM({k|DENUM({b|b∈(e%
e"k)∧pair(p"(ee"k),b)∈R})}) THEN {k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b%
)∈R})} ELSE {k|¬DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}))⊂B∨EDGESET(R%
NG({b|∃k.b=opair(k,p"(ee"k))}) | IF DENUM({k|DENUM({b|b∈(ee"k)∧pair(p"%
(ee"k),b)∈R})}) THEN {k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})} ELSE %
{k|¬DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}))⊂R)

*****∀E L184 {b|∃k.b=opair(k,p"(ee"k))}
*           IF DENUM({k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})})
*              THEN {k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}
*              ELSE {k|¬DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}  ;

269 FNC({b|∃k.b=opair(k,p"(ee"k))})⊃RNG({b|∃k.b=opair(k,p"(ee"k))}) | %
IF DENUM({k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}) THEN {k|DENUM({b%
|b∈(ee"k)∧pair(p"(ee"k),b)∈R})} ELSE {k|¬DENUM({b|b∈(ee"k)∧pair(p"(ee%
"k),b)∈R})})⊂RNG({b|∃k.b=opair(k,p"(ee"k))})

*****∀E AUX23 ↑:*2*1 ↑:*2*2 G;

270 (RNG({b|∃k.b=opair(k,p"(ee"k))}) | IF DENUM({k|DENUM({b|b∈(ee"k)∧p%
air(p"(ee"k),b)∈R})}) THEN {k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}%
 ELSE {k|¬DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})})⊂RNG({b|∃k.b=opair(%
k,p"(ee"k))})∧RNG({b|∃k.b=opair(k,p"(ee"k))})⊂G)⊃RNG({b|∃k.b=opair(k,%
p"(ee"k))}) | IF DENUM({k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}) THE%
N {k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})} ELSE {k|¬DENUM({b|b∈(ee"%
k)∧pair(p"(ee"k),b)∈R})})⊂G

*****TRY 1 USING TAUT 48 ↑↑↑;;

271 RNG({b|∃k.b=opair(k,p"(ee"k))}) | IF DENUM({k|DENUM({b|b∈(ee"k)∧pa%
ir(p"(ee"k),b)∈R})}) THEN {k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})} %
ELSE {k|¬DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})})⊂G  (1 13 17)

*****TRY USING ∧I;
```

```
Goal *1*1*1*1*2*1: DENUM(RNG({b|∃k.b=opair(k,p"(ee"k))} | IF DENUM({%
k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}) THEN {k|DENUM({b|b∈(ee"k)∧%
pair(p"(ee"k),b)∈R})} ELSE {k|¬DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})%
}))
Goal *1*1*1*1*2*2: EDGESET(RNG({b|∃k.b=opair(k,p"(ee"k))} | IF DENUM%
({k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}) THEN {k|DENUM({b|b∈(ee"k%
)∧pair(p"(ee"k),b)∈R})} ELSE {k|¬DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R%
}))))⊂B∨EDGESET(RNG({b|∃k.b=opair(k,p"(ee"k))} | IF DENUM({k|DENUM({b%
|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}) THEN {k|DENUM({b|b∈(ee"k)∧pair(p"(ee%
"k),b)∈R})} ELSE {k|¬DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})))⊂R

*****∀E L95 ↑:*1*1*1 ↑:*1*1*2;

272 FNC({b|∃k.b=opair(k,p"(ee"k))})⊃(IF DENUM({k|DENUM({b|b∈(ee"k)∧pa%
ir(p"(ee"k),b)∈R})}) THEN {k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})} %
ELSE {k|¬DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}⊂DOM({b|∃k.b=opair(k,%
p"(ee"k))})⊃DOM({b|∃k.b=opair(k,p"(ee"k))} | IF DENUM({k|DENUM({b|b∈(%
ee"k)∧pair(p"(ee"k),b)∈R})}) THEN {k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),%
b)∈R})} ELSE {k|¬DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})})=IF DENUM({k%
|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}) THEN {k|DENUM({b|b∈(ee"k)∧p%
air(p"(ee"k),b)∈R})} ELSE {k|¬DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})})

*****TAUT DENUM(↑↑:*1*1*2) 31;

273 DENUM(IF DENUM({k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}) THEN {%
k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})} ELSE {k|¬DENUM({b|b∈(ee"k)∧%
pair(p"(ee"k),b)∈R})})

*****∀E AUX35 30:*2*1 30:*2*2;

274 UNIVERSAL({k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})})⊃(UNIVERSAL(%
{k|¬DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})})⊃({k|DENUM({b|b∈(ee"k)∧pa%
ir(p"(ee"k),b)∈R})}⊂({k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}∪{k|¬D%
ENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})})∧{k|¬DENUM({b|b∈(ee"k)∧pair(p"%
(ee"k),b)∈R})}⊂({k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}∪{k|¬DENUM(%
{b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})})))

*****EVAL ↑;

275 {k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}⊂({k|DENUM({b|b∈(ee"k)∧%
pair(p"(ee"k),b)∈R})}∪{k|¬DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})})∧{k%
|¬DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}⊂({k|DENUM({b|b∈(ee"k)∧pair(%
p"(ee"k),b)∈R})}∪{k|¬DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})})

*****∀E L153 48:*1 ↑↑↑:*1;

276 FNC({b|∃k.b=opair(k,p"(ee"k))})⊃(FNC(CONV({b|∃k.b=opair(k,p"(ee"k%
)})))⊃FNC(CONV({b|∃k.b=opair(k,p"(ee"k))} | IF DENUM({k|DENUM({b|b∈(e%
e"k)∧pair(p"(ee"k),b)∈R})}) THEN {k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b%
)∈R})} ELSE {k|¬DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})})))

*****∀E L161 48:*1 | ↑↑↑↑:*1;

277 FNC({b|∃k.b=opair(k,p"(ee"k))} | IF DENUM({k|DENUM({b|b∈(ee"k)∧pa%
ir(p"(ee"k),b)∈R})}) THEN {k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})} %
```

```
ELSE {k|¬DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}⊃(FNC(CONV({b|∃k.b=o%
pair(k,p"(ee"k))} | IF DENUM({k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R}%
)}) THEN {k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})} ELSE {k|¬DENUM({b%
|b∈(ee"k)∧pair(p"(ee"k),b)∈R})})⊃CONG(DOM({b|∃k.b=opair(k,p"(ee"k))}%
 | IF DENUM({k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}) THEN {k|DENUM%
({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})} ELSE {k|¬DENUM({b|b∈(ee"k)∧pair(p"%
(ee"k),b)∈R})}),RNG({b|∃k.b=opair(k,p"(ee"k))} | IF DENUM({k|DENUM({b%
|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}) THEN {k|DENUM({b|b∈(ee"k)∧pair(p"(ee%
"k),b)∈R})} ELSE {k|¬DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}))))

*****VE AUX34 271:*1*1*2 271:*1;

278 (DENUM(IF DENUM({k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}) THEN %
{k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})} ELSE {k|¬DENUM({b|b∈(ee"k)%
∧pair(p"(ee"k),b)∈R})})∧CONG(IF DENUM({k|DENUM({b|b∈(ee"k)∧pair(p"(ee%
"k),b)∈R})}) THEN {k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})} ELSE {k|%
¬DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})},RNG({b|∃k.b=opair(k,p"(ee"k)%
)} | IF DENUM({k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}) THEN {k|DEN%
UM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})} ELSE {k|¬DENUM({b|b∈(ee"k)∧pair(%
p"(ee"k),b)∈R})})))⊃DENUM(RNG({b|∃k.b=opair(k,p"(ee"k))} | IF DENUM({%
k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}) THEN {k|DENUM({b|b∈(ee"k)∧%
pair(p"(ee"k),b)∈R})} ELSE {k|¬DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}))

*****VE L41 48:*1 273:*1;

279 FNC({b|∃k.b=opair(k,p"(ee"k))})⊃FNC({b|∃k.b=opair(k,p"(ee"k))} | %
IF DENUM({k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}) THEN {k|DENUM({b|b∈%
(ee"k)∧pair(p"(ee"k),b)∈R})} ELSE {k|¬DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})})

*****REWRITE ↑ BY {48}uLOGICTREE;

2 substitutions were made

280 FNC({b|∃k.b=opair(k,p"(ee"k))}) | IF DENUM({k|DENUM({b|b∈(ee"k)∧pa%
ir(p"(ee"k),b)∈R})}) THEN {k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})} %
ELSE {k|¬DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})})

*****TAUT 268:*1 268;

281 FNC(CONV({b|∃k.b=opair(k,p"(ee"k))})) (1 13 17)

*****TAUT 268:*2*1 268;

282 DOM({b|∃k.b=opair(k,p"(ee"k))})=omega (1 13 17)

*****REWRITE 276 BY {48 ↑↑}uLOGICTREE;

4 substitutions were made

283 FNC(CONV({b|∃k.b=opair(k,p"(ee"k))}) | IF DENUM({k|DENUM({b|b∈(ee"%
k)∧pair(p"(ee"k),b)∈R})}) THEN {k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈%
R})} ELSE {k|¬DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})})) (1 13 17)

*****REWRITE 277 BY {280 ↑}uLOGICTREE;
```

4 substitutions were made

284 CONG(DOM({b|∃k.b=opair(k,p"(ee"k))}) | IF DENUM({k|DENUM({b|b∈(ee"%
k)∧pair(p"(ee"k),b)∈R})}) THEN {k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈%
R})} ELSE {k|¬DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}),RNG({b|∃k.b=op%
air(k,p"(ee"k))}) | IF DENUM({k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})%
}) THEN {k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})} ELSE {k|¬DENUM({b|%
b∈(ee"k)∧pair(p"(ee"k),b)∈R})})) (1 13 17)

*****SUBSTR 30 IN 275;

285 {κ|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}⊂omega∧{k|¬DENUM({b|b∈(%
ee"k)∧pair(p"(ee"k),b)∈R})}⊂omega

*****TAUT ↑:*1 ↑;

286 {k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}⊂omega

*****TAUT ↑↑:*2 ↑↑;

287 {k|¬DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}⊂omega

*****REWRITE 272:*2*1 BY {282 ↑↑,↑}∪LOGICTREE∪ARGIFTREE;

9 substitutions were made

288 IF DENUM({k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}) THEN {k|DENU%
M({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})} ELSE {k|¬DENUM({b|b∈(ee"k)∧pair(p%
"(ee"k),b)∈R})}⊂DOM({b|∃k.b=opair(k,p"(ee"k))}) (1 13 17)

*****REWRITE 272 BY {48 ↑}∪LOGICTREE;

4 substitutions were made

289 DOM({b|∃k.b=opair(k,p"(ee"k))}) | IF DENUM({k|DENUM({b|b∈(ee"k)∧pa%
ir(p"(ee"k),b)∈R})}) THEN {k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})} %
ELSE {k|¬DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}=IF DENUM({k|DENUM({%
b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}) THEN {k|DENUM({b|b∈(ee"k)∧pair(p"(e%
e"k),b)∈R})} ELSE {k|¬DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})} (1 13 17)

*****REWRITE 284 BY {↑};

1 substitutions were made

290 CONG(IF DENUM({k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}) THEN {k%
|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})} ELSE {k|¬DENUM({b|b∈(ee"k)∧p%
air(p"(ee"k),b)∈R})},RNG({b|∃k.b=opair(k,p"(ee"k))}) | IF DENUM({k|DEN%
UM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}) THEN {k|DENUM({b|b∈(ee"k)∧pair(%
p"(ee"k),b)∈R})} ELSE {k|¬DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})})) (1 13 17)

*****REWRITE 278 BY {273 ↑}∪LOGICTREE;

4 substitutions were made

```
291 DENUM(RNG({b|∃k.b=opair(k,p"(ee"k))}) | IF DENUM({k|DENUM({b|b∈(ee%
"k)∧pair(p"(ee"k),b)∈R})}) THEN {k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)%
∈R})} ELSE {k|¬DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}))  (1 13 17)

*****QED *1*1*1*1*2*1;

*****∀E RNG {b|∃k.b=opair(k,p"(ee"k))} |
*              IF DENUM({k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})})
*                 THEN {k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}
*                 ELSE {k|¬DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})} ;

292 FNC({b|∃k.b=opair(k,p"(ee"k))} | IF DENUM({k|DENUM({b|b∈(ee"k)∧pa%
ir(p"(ee"k),b)∈R})}) THEN {k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})} %
ELSE {k|¬DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})})⊃RNG({b|∃k.b=opair(k%
,p"(ee"k))} | IF DENUM({k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}) TH%
EN {k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})} ELSE {k|¬DENUM({b|b∈(ee%
"k)∧pair(p"(ee"k),b)∈R})})={c|∃a.opair(a,c)∈({b|∃k.b=opair(k,p"(ee"k)%
)} | IF DENUM({k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}) THEN {k|DEN%
UM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})} ELSE {k|¬DENUM({b|b∈(ee"k)∧pair(%
p"(ee"k),b)∈R})})}

*****∀E L41 ↑:*1*1*1 ↑:*1*1*2;

293 FNC({b|∃k.b=opair(k,p"(ee"k))})⊃FNC({b|∃k.b=opair(k,p"(ee"k))} | IF %
DENUM({k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}) THEN {k|DENUM({b|b∈%
(ee"k)∧pair(p"(ee"k),b)∈R})} ELSE {k|¬DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})})

*****∀E RESTR ↑↑:*1*1*1 ↑↑:*1*1*2;

294 FNC({b|∃k.b=opair(k,p"(ee"k))})⊃({b|∃k.b=opair(k,p"(ee"k))} | IF %
DENUM({k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}) THEN {k|DENUM({b|b∈%
(ee"k)∧pair(p"(ee"k),b)∈R})} ELSE {k|¬DENUM({b|b∈(ee"k)∧pair(p"(ee"k)%
,b)∈R})})=({b|∃k.b=opair(k,p"(ee"k))}∩CROSS(IF DENUM({k|DENUM({b|b∈(e%
e"k)∧pair(p"(ee"k),b)∈R})}) THEN {k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b%
)∈R})} ELSE {k|¬DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})},V))

*****REWRITE ↑↑ BY {48}uLOGICTREE;

2 substitutions were made

295 FNC({b|∃k.b=opair(k,p"(ee"k))} | IF DENUM({k|DENUM({b|b∈(ee"k)∧pa%
ir(p"(ee"k),b)∈R})}) THEN {k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})} %
ELSE {k|¬DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})})

*****REWRITE ↑↑ BY {48}uLOGICTREE;

2 substitutions were made

296 ({b|∃k.b=opair(k,p"(ee"k))} | IF DENUM({k|DENUM({b|b∈(ee"k)∧pair(%
p"(ee"k),b)∈R})}) THEN {k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})} ELS%
E {k|¬DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})})=({b|∃k.b=opair(k,p"(ee%
"k))}∩CROSS(IF DENUM({k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}) THEN%
 {k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})} ELSE {k|¬DENUM({b|b∈(ee"k%
)∧pair(p"(ee"k),b)∈R})},V))
```

```
*****REWRITE ↑↑↑↑↑ BY {↑↑}∪LOGICTREE;

2 substitutions were made

297 RNG({b|∃k.b=opair(k,p"(ee"k))} | IF DENUM({k|DENUM({b|b∈(ee"k)∧pa%
ir(p"(ee"k),b)∈R})}) THEN {k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})} %
ELSE {k|¬DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})})={c|∃a.opair(a,c)∈({%
b|∃k.b=opair(k,p"(ee"k))} | IF DENUM({k|DENUM({b|b∈(ee"k)∧pair(p"(ee"%
k),b)∈R})}) THEN {k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})} ELSE {k|¬%
DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})})}

*****TRY USING REWRITE BY {↑,↑↑};

Goal #1#1#1#1#2#2#1: EDGESET({c|∃a.opair(a,c)∈({b|∃k.b=opair(k,p"(ee%
"k))}∩CROSS(IF DENUM({k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}) THEN%
 {k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})} ELSE {k|¬DENUM({b|b∈(ee"k%
)∧pair(p"(ee"k),b)∈R})},V))})⊂B∨EDGESET({c|∃a.opair(a,c)∈({b|∃k.b=opa%
ir(k,p"(ee"k))}∩CROSS(IF DENUM({k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈%
R})}) THEN {k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})} ELSE {k|¬DENUM(%
{b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})},V))})⊂R

*****TRY USING IFCASES DENUM({k|DENUM({b|b∈ee"k∧pair(p"(ee"k),b)∈R})});

Goal #1#1#1#1#2#2#1#1: DENUM({k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R%
})})⊃(EDGESET({c|∃a.opair(a,c)∈({b|∃k.b=opair(k,p"(ee"k))}∩CROSS(IF D%
ENUM({k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}) THEN {k|DENUM({b|b∈(%
ee"k)∧pair(p"(ee"k),b)∈R})} ELSE {k|¬DENUM({b|b∈(ee"k)∧pair(p"(ee"k),%
b)∈R})},V))})⊂B∨EDGESET({c|∃a.opair(a,c)∈({b|∃k.b=opair(k,p"(ee"k))}∩%
CROSS(IF DENUM({k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}) THEN {k|DE%
NUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})} ELSE {k|¬DENUM({b|b∈(ee"k)∧pair%
(p"(ee"k),b)∈R})},V))})⊂R)
Goal #1#1#1#1#2#2#1#2: ¬DENUM({k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈%
R})})⊃(EDGESET({c|∃a.opair(a,c)∈({b|∃k.b=opair(k,p"(ee"k))}∩CROSS(IF %
DENUM({k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}) THEN {k|DENUM({b|b∈%
(ee"k)∧pair(p"(ee"k),b)∈R})} ELSE {k|¬DENUM({b|b∈(ee"k)∧pair(p"(ee"k)%
,b)∈R})},V))})⊂B∨EDGESET({c|∃a.opair(a,c)∈({b|∃k.b=opair(k,p"(ee"k))}%
∩CROSS(IF DENUM({k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}) THEN {k|D%
ENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})} ELSE {k|¬DENUM({b|b∈(ee"k)∧pai%
r(p"(ee"k),b)∈R})},V))})⊂R)
Goal #1#1#1#1#2#2#1#1#1: EDGESET({c|∃a.opair(a,c)∈({b|∃k.b=opair(k,p%
"(ee"k))}∩CROSS(IF DENUM({k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}) %
THEN {k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})} ELSE {k|¬DENUM({b|b∈(%
ee"k)∧pair(p"(ee"k),b)∈R})},V))})⊂B∨EDGESET({c|∃a.opair(a,c)∈({b|∃k.b%
=opair(k,p"(ee"k))}∩CROSS(IF DENUM({k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k)%
,b)∈R})}) THEN {k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})} ELSE {k|¬DE%
NUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})},V))})⊂R
298 DENUM({k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})})  (298)

Goal #1#1#1#1#2#2#1#1#1#1: EDGESET({c|∃a.opair(a,c)∈({b|∃k.b=opair(k%
,p"(ee"k))}∩CROSS({k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})},V))})⊂B∨%
EDGESET({c|∃a.opair(a,c)∈({b|∃k.b=opair(k,p"(ee"k))}∩CROSS({k|DENUM({%
b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})},V))})⊂R
Goal #1#1#1#1#2#2#1#2#1: EDGESET({c|∃a.opair(a,c)∈({b|∃k.b=opair(k,p%
"(ee"k))}∩CROSS(IF DENUM({k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}) %
THEN {k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})} ELSE {k|¬DENUM({b|b∈(%
```

```
ee"k)∧pair(p"(ee"k),b)∈R)}},V)}})⊂B∨EDGESET({c|∃a.opair(a,c)∈({b|∃k.b%
=opair(k,p"(ee"k))}∩CROSS(IF DENUM({k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k)%
,b)∈R})}) THEN {k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})} ELSE {k|¬DE%
NUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})},V)})}⊂R
299 ¬DENUM({k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})})  (299)

300 DENUM({k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})})≠FALSE  (299)

Goal *1*1*1*1*2*2*1*2*1*1: EDGESET({c|∃a.opair(a,c)∈({b|∃k.b=opair(k%
,p"(ee"k))}∩CROSS({k|¬DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})},V)})}⊂B%
∨EDGESET({c|∃a.opair(a,c)∈({b|∃k.b=opair(k,p"(ee"k))}∩CROSS({k|¬DENUM%
({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})},V)})}⊂R

*****TRY USING ORI 1;

Goal *1*1*1*1*2*2*1*2*1*1*1: EDGESET({c|∃a.opair(a,c)∈({b|∃k.b=opair%
(k,p"(ee"k))}∩CROSS({k|¬DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})},V)})}⊂B

*****TRY USING REWRITE BY {EDGESET AUX27 AUX25 CROSS V SUBSET AUX5};

Goal *1*1*1*1*2*2*1*2*1*1*1*1: ∀c1.((SET(c1)∧∃c d1.((SET(c)∧∃a.(((SE%
T(a)∧SET(c))∧∃k.(a=k∧c=(p"(ee"k))))∧((SET(a)∧SET(c))∧∃d e.((a=d∧c=e)∧%
((NATNUM(d)∧¬DENUM({b|b∈(ee"d)∧pair(p"(ee"d),b)∈R}))∧SET(e))))))∧((SE%
T(d1)∧∃a.(((SET(a)∧SET(d1))∧∃k.(a=k∧d1=(p"(ee"k))))∧((SET(a)∧SET(d1))%
∧∃d e.((a=d∧d1=e)∧((NATNUM(d)∧¬DENUM({b|b∈(ee"d)∧pair(p"(ee"d),b)∈R})%
)∧SET(e))))))∧(¬(c=d1)∧c1=pair(c,d1))))))⊃c1∈B)

*****TRY *1*1*1*1*2*2*1*1*1*1 USING ORI 2;

Goal *1*1*1*1*2*2*1*1*1*1*1: EDGESET({c|∃a.opair(a,c)∈({b|∃k.b=opair%
(k,p"(ee"k))}∩CROSS({k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})},V)})}⊂R

*****TRY USING REWRITE BY {EDGESET AUX27 AUX25 CROSS V SUBSET AUX5};

Goal *1*1*1*1*2*2*1*1*1*1*1*1: ∀c1.((SET(c1)∧∃c d1.((SET(c)∧∃a.(((SE%
T(a)∧SET(c))∧∃k.(a=k∧c=(p"(ee"k))))∧((SET(a)∧SET(c))∧∃d e.((a=d∧c=e)∧%
((NATNUM(d)∧DENUM({b|b∈(ee"d)∧pair(p"(ee"d),b)∈R}))∧SET(e))))))∧((SET%
(d1)∧∃a.(((SET(a)∧SET(d1))∧∃k.(a=k∧d1=(p"(ee"k))))∧((SET(a)∧SET(d1))∧%
∃d e.((a=d∧d1=e)∧((NATNUM(d)∧DENUM({b|b∈(ee"d)∧pair(p"(ee"d),b)∈R}))∧%
SET(e))))))∧(¬(c=d1)∧c1=pair(c,d1))))))⊃c1∈R)

*****GOAL ∀i j.(i LT j⊃IF DENUM({b|b∈ee"i∧pair(p"(ee"i),b)∈R})
* THEN pair(p"(ee"i),p"(ee"j))∈R
* ELSE pair(p"(ee"i),p"(ee"j))∈B );

Goal *14:  ∀i j.(i LT j⊃IF DENUM({b|b∈(ee"i)∧pair(p"(ee"i),b)∈R}) THEN %
pair(p"(ee"i),p"(ee"j))∈R ELSE pair(p"(ee"i),p"(ee"j))∈B)

*****TRY USING ELIMINATION;

Goal *14*1:  i LT j⊃IF DENUM({b|b∈(ee"i)∧pair(p"(ee"i),b)∈R}) THEN pa%
ir(p"(ee"i),p"(ee"j))∈R ELSE pair(p"(ee"i),p"(ee"j))∈B
Goal *14*1*1:  IF DENUM({b|b∈(ee"i)∧pair(p"(ee"i),b)∈R}) THEN pair(p"%
(ee"i),p"(ee"j))∈R ELSE pair(p"(ee"i),p"(ee"j))∈B
```

*****PREPARE;

301 i LT j (301)

*****VE 165 i j;

302 i LT j⊃(ee"j)⊂(ee"SUC(i)) (1 13 17)

*****TAUT ↑:#2 ↑↑;

303 (ee"j)⊂(ee"SUC(i)) (1 13 17 301)

*****REWRITE ↑ BY {SUBSET SUCI};

2 substitutions were made

304 ∀c.(c∈(ee"j)⊃c∈IF DENUM({b|b∈(ee"i)∧pair(p"(ee"i),b)∈R}) THEN {b|%
b∈(ee"i)∧pair(p"(ee"i),b)∈R} ELSE {b|b∈(ee"i)∧pair(p"(ee"i),b)∈B}) (1 13 17 301)

*****VE ↑ p"(ee"j);

305 (p"(ee"j))∈(ee"j)⊃(p"(ee"j))∈IF DENUM({b|b∈(ee"i)∧pair(p"(ee"i),b%
)∈R}) THEN {b|b∈(ee"i)∧pair(p"(ee"i),b)∈R} ELSE {b|b∈(ee"i)∧pair(p"(e%
e"i),b)∈B} (1 13 17 301)

*****REWRITE ↑ BY {PINEE}∪COMPTREE∪LOGICTREE∪ARGIFTREE∪WFFIFTREE;

6 substitutions were made

306 (DENUM({b|b∈(ee"i)∧pair(p"(ee"i),b)∈R})⊃(SET(p"(ee"j))∧((p"(ee"j))%
∈(ee"i)∧pair(p"(ee"i),p"(ee"j))∈R)))∧(¬DENUM({b|b∈(ee"i)∧pair(p"(ee"i),b)%
∈R})⊃(SET(p"(ee"j))∧((p"(ee"j))∈(ee"i)∧pair(p"(ee"i),p"(ee"j))∈B))) (1 13 17 301)

*****TRY USING TAUT ↑;

307 IF DENUM({b|b∈(ee"i)∧pair(p"(ee"i),b)∈R}) THEN pair(p"(ee"i),p"(e%
e"j))∈R ELSE pair(p"(ee"i),p"(ee"j))∈B (1 13 17 301)

308 i LT j⊃IF DENUM({b|b∈(ee"i)∧pair(p"(ee"i),b)∈R}) THEN pair(p"(ee"%
i),p"(ee"j))∈R ELSE pair(p"(ee"i),p"(ee"j))∈B (1 13 17)

309 ∀i j.(i LT j⊃IF DENUM({b|b∈(ee"i)∧pair(p"(ee"i),b)∈R}) THEN pair(%
p"(ee"i),p"(ee"j))∈R ELSE pair(p"(ee"i),p"(ee"j))∈B) (1 13 17)

*****TRY #1#1#1#1#2#2#1#1#1#1#1#1 USING ELIMINATION;

Goal #1#1#1#1#2#2#1#1#1#1#1#1: (SET(c1)∧∃c d1.((SET(c)∧∃a.(((SET(a%
)∧SET(c))∧∃k.(a=k∧c=(p"(ee"k))))∧((SET(a)∧SET(c))∧∃d e.((a=d∧c=e)∧((N%
ATNUM(d)∧DENUM({b|b∈(ee"d)∧pair(p"(ee"d),b)∈R}))∧SET(e))))))∧((SET(d1%
)∧∃a.(((SET(a)∧SET(d1))∧∃k.(a=k∧d1=(p"(ee"k))))∧((SET(a)∧SET(d1))∧∃d %
e.((a=d∧d1=e)∧((NATNUM(d)∧DENUM({b|b∈(ee"d)∧pair(p"(ee"d),b)∈R}))∧SET%
(e))))))∧(¬(c=d1)∧c1=pair(c,d1)))))⊃c1∈R
Goal #1#1#1#1#2#2#1#1#1#1#1#1#1#1: c1∈R

*****PREPARE;

310 SET(c1)∧∃c d1.((SET(c)∧∃a.(((SET(a)∧SET(c))∧∃k.(a=k∧c=(p"(ee"k)))%
)∧((SET(a)∧SET(c))∧∃d e.((a=d∧c=e)∧((NATNUM(d)∧DENUM({b|b∈(ee"d)∧pair%
(p"(ee"d),b)∈R}))∧SET(e))))))∧((SET(d1)∧∃a.(((SET(a)∧SET(d1))∧∃k.(a=k%
∧d1=(p"(ee"k))))∧((SET(a)∧SET(d1))∧∃d e.((a=d∧d1=e)∧((NATNUM(d)∧DENUM%
({b|b∈(ee"d)∧pair(p"(ee"d),b)∈R}))∧SET(e))))))∧(¬(c=d1)∧c1=pair(c,d1)))) (310)

311 ∃c d1.((SET(c)∧∃a.(((SET(a)∧SET(c))∧∃k.(a=k∧c=(p"(ee"k))))∧((SET(%
a)∧SET(c))∧∃d e.((a=d∧c=e)∧((NATNUM(d)∧DENUM({b|b∈(ee"d)∧pair(p"(ee"d%
),b)∈R}))∧SET(e))))))∧((SET(d1)∧∃a.(((SET(a)∧SET(d1))∧∃k.(a=k∧d1=(p"(%
ee"k))))∧((SET(a)∧SET(d1))∧∃d e.((a=d∧d1=e)∧((NATNUM(d)∧DENUM({b|b∈(e%
e"d)∧pair(p"(ee"d),b)∈R}))∧SET(e))))))∧(¬(c=d1)∧c1=pair(c,d1)))) (310)

312 SET(c1) (310)

*****ES ↑↑ c d1;

313 (SET(c)∧∃a.(((SET(a)∧SET(c))∧∃k.(a=k∧c=(p"(ee"k))))∧((SET(a)∧SET(%
c))∧∃d e.((a=d∧c=e)∧((NATNUM(d)∧DENUM({b|b∈(ee"d)∧pair(p"(ee"d),b)∈R}%
))∧SET(e))))))∧((SET(d1)∧∃a.(((SET(a)∧SET(d1))∧∃k.(a=k∧d1=(p"(ee"k)))%
)∧((SET(a)∧SET(d1))∧∃d e.((a=d∧d1=e)∧((NATNUM(d)∧DENUM({b|b∈(ee"d)∧pa%
ir(p"(ee"d),b)∈R}))∧SET(e))))))∧(¬(c=d1)∧c1=pair(c,d1))) (313)

*****ADDFACTS #1#1#1#1#2#2#1#1#1#1#1#1#1 ASSUME ↑;

Goal #1#1#1#1#2#2#1#1#1#1#1#1#1: c1∈R

*****PREPARE;

314 c1=pair(c,d1) (313)

315 ¬(c=d1) (313)

316 ∃a.(((SET(a)∧SET(d1))∧∃k.(a=k∧d1=(p"(ee"k))))∧((SET(a)∧SET(d1))∧∃d e.%
((a=d∧d1=e)∧((NATNUM(d)∧DENUM({b|b∈(ee"d)∧pair(p"(ee"d),b)∈R}))∧SET(e))))) (313)

317 SET(d1) (313)

318 ∃a.(((SET(a)∧SET(c))∧∃k.(a=k∧c=(p"(ee"k))))∧((SET(a)∧SET(c))∧∃d e.%
((a=d∧c=e)∧((NATNUM(d)∧DENUM({b|b∈(ee"d)∧pair(p"(ee"d),b)∈R}))∧SET(e))))) (313)

319 SET(c) (313)

320 c=d1=FALSE (313)

*****ES 316 a;

321 ((SET(a)∧SET(d1))∧∃k.(a=k∧d1=(p"(ee"k))))∧((SET(a)∧SET(d1))∧∃d e.%
((a=d∧d1=e)∧((NATNUM(d)∧DENUM({b|b∈(ee"d)∧pair(p"(ee"d),b)∈R}))∧SET(e)))) (321)

*****ES 318 a1;

322 ((SET(a1)∧SET(c))∧∃k.(a1=k∧c=(p"(ee"k))))∧((SET(a1)∧SET(c))∧∃d e.%

((a1=d∧c=e)∧((NATNUM(d)∧DENUM({b|b∈(ee"d)∧pair(p"(ee"d),b)∈R}))∧SET(e)))) (322)

*****ADDFACTS *1*1*1*1*2*2*1*1*1*1*1*1*1*1 ASSUME ↑,↑↑;

Goal *1*1*1*1*2*2*1*1*1*1*1*1*1*1: c1∈R

*****PREPARE;

323 ∃d e.((a1=d∧c=e)∧((NATNUM(d)∧DENUM({b|b∈(ee"d)∧pair(p"(ee"d),b)∈R}))∧SET(e))) (322)

324 SET(c) (313)

325 SET(a1) (310)

326 ∃k.(a1=k∧c=(p"(ee"k))) (310)

327 SET(c) (310)

328 SET(a1) (310)

329 ∃d e.((a=d∧d1=e)∧((NATNUM(d)∧DENUM({b|b∈(ee"d)∧pair(p"(ee"d),b)∈R}))∧SET(e))) (321)

330 SET(d1) (313)

331 SET(a) (310)

332 ∃k.(a=k∧d1=(p"(ee"k))) (310)

333 SET(d1) (310)

334 SET(a) (310)

*****ES 323 d e;

335 (a1=d∧c=e)∧((NATNUM(d)∧DENUM({b|b∈(ee"d)∧pair(p"(ee"d),b)∈R}))∧SET(e)) (335)

*****ES 326 k;

336 a1=k∧c=(p"(ee"k)) (336)

*****ES 329 d2 e2;

337 (a=d2∧a1=e2)∧((NATNUM(d2)∧DENUM({b|b∈(ee"d2)∧pair(p"(ee"d2),b)∈R}))∧SET(e2)) (337)
*****ES 332 j;

338 a=j∧d1=(p"(ee"j)) (338)

*****TAUTEQ c=p"(ee"k) 335;;

339 c=(p"(ee"k)) (310 313 322 336)

*****TAUTEQ d1=p"(ee"j) 335;;

```
340 d1=(p"(ee"j))  (310 313 321 338)

*****TAUTEQ d2=j  335;;

341 d2=j  (310 313 321 337 338)

*****TAUTEQ d=k  335;;

342 d=k  (310 313 322 335 336)

*****TRY USING REWRITE BY {314 ↑↑↑↑,↑↑↑};

Goal *1*1*1*1*1*2*2*1*1*1*1*1*1*1*1*1*1*1:  pair(p"(ee"k),p"(ee"j))∈R

*****ASSUME k=j;

343 k=j  (343)

*****REWRITE 339 BY {↑};

1 substitutions were made

344 c=(p"(ee"j))  (310 313 322 336 343)

*****TAUTEQ FALSE 315 340 ↑;

345 FALSE  (310 313 321 322 336 338 343)

*****¬I ↑ 343;

346 ¬(k=j)  (310 313 321 322 336 338)

*****REWRITE 335 BY {342};

4 substitutions were made

347 (a1=k∧c=e)∧((NATNUM(k)∧DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R}))∧SE%
T(e))  (310 313 322 335 336)

*****REWRITE 337 BY {341};

4 substitutions were made

348 (a=j∧d1=e2)∧((NATNUM(j)∧DENUM({b|b∈(ee"j)∧pair(p"(ee"j),b)∈R}))∧S%
ET(e2))  (310 313 321 337 338)

*****REWRITE ↑↑↑ BY {LESS2};

1 substitutions were made

349 k LT j∨j LT k  (310 313 321 322 336 338)

*****∀E 309 k j;
```

```
350 k LT j⊃IF DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R}) THEN pair(p"(ee"%
k),p"(ee"j))∈R ELSE pair(p"(ee"k),p"(ee"j))∈B  (1 13 17)

*****∀E 309 j k;

351 j LT k⊃IF DENUM({b|b∈(ee"j)∧pair(p"(ee"j),b)∈R}) THEN pair(p"(ee"%
j),p"(ee"k))∈R ELSE pair(p"(ee"j),p"(ee"k))∈B  (1 13 17)

*****∀E AUX24 p"(ee"k) p"(ee"j);

352 pair(p"(ee"k),p"(ee"j))=pair(p"(ee"j),p"(ee"k))

*****TAUTEQ ↑:#1∈R ↑↑↑↑↑↑;;

353 pair(p"(ee"k),p"(ee"j))∈R  (1 13 17 310 313 321 322 336 338)

*****QED;

354 c1∈R≡pair(p"(ee"k),p"(ee"j))∈R  (310 313 321 322 336 338)

355 c1∈R  (1 13 17 310)

356 (SET(c1)∧∃c d1.((SET(c)∧∃a.(((SET(a)∧SET(c))∧∃k.(a=k∧c=(p"(ee"k))%
))∧((SET(a)∧SET(c))∧∃d e.((a=d∧c=e)∧((NATNUM(d)∧DENUM({b|b∈(ee"d)∧pai%
r(p"(ee"d),b)∈R}))∧SET(e))))))∧((SET(d1)∧∃a.(((SET(a)∧SET(d1))∧∃k.(a=%
k∧d1=(p"(ee"k))))∧((SET(a)∧SET(d1))∧∃d e.((a=d∧d1=e)∧((NATNUM(d)∧DENU%
M({b|b∈(ee"d)∧pair(p"(ee"d),b)∈R}))∧SET(e))))))∧(¬(c=d1)∧c1=pair(c,d1%
)))))⊃c1∈R  (1 13 17)

357 ∀c1.((SET(c1)∧∃c d1.((SET(c)∧∃a.(((SET(a)∧SET(c))∧∃k.(a=k∧c=(p"(e%
e"k))))∧((SET(a)∧SET(c))∧∃d e.((a=d∧c=e)∧((NATNUM(d)∧DENUM({b|b∈(ee"d%
)∧pair(p"(ee"d),b)∈R}))∧SET(e))))))∧((SET(d1)∧∃a.(((SET(a)∧SET(d1))∧∃%
k.(a=k∧d1=(p"(ee"k))))∧((SET(a)∧SET(d1))∧∃d e.((a=d∧d1=e)∧((NATNUM(d)%
∧DENUM({b|b∈(ee"d)∧pair(p"(ee"d),b)∈R}))∧SET(e))))))∧(¬(c=d1)∧c1=pair%
(c,d1)))))⊃c1∈R)  (1 13 17)

358 EDGESET({c|∃a.opair(a,c)∈({b|∃k.b=opair(k,p"(ee"k))}∩CROSS({k|DEN%
UM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})},V))})⊂R≡∀c1.((SET(c1)∧∃c d1.((SE%
T(c)∧∃a.(((SET(a)∧SET(c))∧∃k.(a=k∧c=(p"(ee"k))))∧((SET(a)∧SET(c))∧∃d %
e.((a=d∧c=e)∧((NATNUM(d)∧DENUM({b|b∈(ee"d)∧pair(p"(ee"d),b)∈R}))∧SET(%
e))))))∧((SET(d1)∧∃a.(((SET(a)∧SET(d1))∧∃k.(a=k∧d1=(p"(ee"k))))∧((SET%
(a)∧SET(d1))∧∃d e.((a=d∧d1=e)∧((NATNUM(d)∧DENUM({b|b∈(ee"d)∧pair(p"(e%
e"d),b)∈R}))∧SET(e))))))∧(¬(c=d1)∧c1=pair(c,d1)))))⊃c1∈R)

359 EDGESET({c|∃a.opair(a,c)∈({b|∃k.b=opair(k,p"(ee"k))}∩CROSS({k|DEN%
UM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})},V))})⊂R  (1 13 17)

360 EDGESET({c|∃a.opair(a,c)∈({b|∃k.b=opair(k,p"(ee"k))}∩CROSS({k|DEN%
UM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})},V))})⊂B∨EDGESET({c|∃a.opair(a,c)%
∈({b|∃k.b=opair(k,p"(ee"k))}∩CROSS({k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k)%
,b)∈R})},V))})⊂R  (1 13 17)

361 (EDGESET({c|∃a.opair(a,c)∈({b|∃k.b=opair(k,p"(ee"k))}∩CROSS(IF DE%
NUM({k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}) THEN {k|DENUM({b|b∈(e%
```

```
e"k)∧pair(p"(ee"k),b)∈R})} ELSE {k|¬DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b%
)∈R})},V)})}⊂B∨EDGESET({c|∃a.opair(a,c)∈({b|∃k.b=opair(k,p"(ee"k))}∩C%
ROSS(IF DENUM({k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}) THEN {k|DEN%
UM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})} ELSE {k|¬DENUM({b|b∈(ee"k)∧pair(%
p"(ee"k),b)∈R})},V)})}⊂R)≡(EDGESET({c|∃a.opair(a,c)∈({b|∃k.b=opair(k,%
p"(ee"k))}∩CROSS({k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})},V)})}⊂B∨E%
DGESET({c|∃a.opair(a,c)∈({b|∃k.b=opair(k,p"(ee"k))}∩CROSS({k|DENUM({b%
|b∈(ee"k)∧pair(p"(ee"k),b)∈R})},V)})}⊂R)  (298)

362 EDGESET({c|∃a.opair(a,c)∈({b|∃k.b=opair(k,p"(ee"k))}∩CROSS(IF DEN%
UM({k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}) THEN {k|DENUM({b|b∈(ee%
"k)∧pair(p"(ee"k),b)∈R})} ELSE {k|¬DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)%
∈R})},V)})}⊂B∨EDGESET({c|∃a.opair(a,c)∈({b|∃k.b=opair(k,p"(ee"k))}∩CR%
OSS(IF DENUM({k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}) THEN {k|DENU%
M({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})} ELSE {k|¬DENUM({b|b∈(ee"k)∧pair(p%
"(ee"k),b)∈R})},V)})}⊂R  (1 13 17 298)

363 DENUM({k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})})⊃(EDGESET({c|∃a.%
opair(a,c)∈({b|∃k.b=opair(k,p"(ee"k))}∩CROSS(IF DENUM({k|DENUM({b|b∈(%
ee"k)∧pair(p"(ee"k),b)∈R})}) THEN {k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),%
b)∈R})} ELSE {k|¬DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})},V)})}⊂B∨EDGE%
SET({c|∃a.opair(a,c)∈({b|∃k.b=opair(k,p"(ee"k))}∩CROSS(IF DENUM({k|DE%
NUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}) THEN {k|DENUM({b|b∈(ee"k)∧pair%
(p"(ee"k),b)∈R})} ELSE {k|¬DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})},V)})}⊂R)  (1 13 17)

*****TRY USING ELIMINATION;

Goal #1#1#1#1#2#2#1#2#1#1#1#1#1: (SET(c1)∧∃c d1.((SET(c)∧∃a.(((SET(a%
)∧SET(c))∧∃k.(a=k∧c=(p"(ee"k))))∧((SET(a)∧SET(c))∧∃d e.((a=d∧c=e)∧((N%
ATNUM(d)∧¬DENUM({b|b∈(ee"d)∧pair(p"(ee"d),b)∈R}))∧SET(e))))))∧((SET(d%
1)∧∃a.(((SET(a)∧SET(d1))∧∃k.(a=k∧d1=(p"(ee"k))))∧((SET(a)∧SET(d1))∧∃d%
 e.((a=d∧d1=e)∧((NATNUM(d)∧¬DENUM({b|b∈(ee"d)∧pair(p"(ee"d),b)∈R}))∧S%
ET(e))))))∧(¬(c=d1)∧c1=pair(c,d1)))))⊃c1∈B
Goal #1#1#1#1#2#2#1#2#1#1#1#1#1#1: c1∈B

*****PREPARE;

364 SET(c1)∧∃c d1.((SET(c)∧∃a.(((SET(a)∧SET(c))∧∃k.(a=k∧c=(p"(ee"k)))%
)∧((SET(a)∧SET(c))∧∃d e.((a=d∧c=e)∧((NATNUM(d)∧¬DENUM({b|b∈(ee"d)∧pai%
r(p"(ee"d),b)∈R}))∧SET(e))))))∧((SET(d1)∧∃a.(((SET(a)∧SET(d1))∧∃k.(a=%
k∧d1=(p"(ee"k))))∧((SET(a)∧SET(d1))∧∃d e.((a=d∧d1=e)∧((NATNUM(d)∧¬DENUM%
({b|b∈(ee"d)∧pair(p"(ee"d),b)∈R}))∧SET(e))))))∧(¬(c=d1)∧c1=pair(c,d1))))  (364)

365 ∃c d1.((SET(c)∧∃a.(((SET(a)∧SET(c))∧∃k.(a=k∧c=(p"(ee"k))))∧((SET(%
a)∧SET(c))∧∃d e.((a=d∧c=e)∧((NATNUM(d)∧¬DENUM({b|b∈(ee"d)∧pair(p"(ee"%
d),b)∈R}))∧SET(e))))))∧((SET(d1)∧∃a.(((SET(a)∧SET(d1))∧∃k.(a=k∧d1=(p"%
(ee"k))))∧((SET(a)∧SET(d1))∧∃d e.((a=d∧d1=e)∧((NATNUM(d)∧¬DENUM({b|b∈%
(ee"d)∧pair(p"(ee"d),b)∈R}))∧SET(e))))))∧(¬(c=d1)∧c1=pair(c,d1))))  (364)

366 SET(c1)  (364)

*****ES ↑↑ c d1;

367 (SET(c)∧∃a.(((SET(a)∧SET(c))∧∃k.(a=k∧c=(p"(ee"k))))∧((SET(a)∧SET(%
c))∧∃d e.((a=d∧c=e)∧((NATNUM(d)∧¬DENUM({b|b∈(ee"d)∧pair(p"(ee"d),b)∈R%
```

}))∧SET(e))))))∧((SET(d1)∧∃a.(((SET(a)∧SET(d1))∧∃k.(a=k∧d1=(p"(ee"k))%
))∧((SET(a)∧SET(d1))∧∃d e.((a=d∧d1=e)∧((NATNUM(d)∧¬DENUM({b|b∈(ee"d)∧%
pair(p"(ee"d),b)∈R}))∧SET(e))))))∧(¬(c=d1)∧c1=pair(c,d1))) (367)

*****ADDFACTS #1#1#1#1#2#2#1#2#1#1#1#1#1#1 ASSUME ↑;

Goal #1#1#1#1#2#2#1#2#1#1#1#1#1#1: c1∈B

*****PREPARE;

368 c1=pair(c,d1) (367)

369 ¬(c=d1) (367)

370 ∃a.(((SET(a)∧SET(d1))∧∃k.(a=k∧d1=(p"(ee"k))))∧((SET(a)∧SET(d1))∧∃d e.%
((a=d∧d1=e)∧((NATNUM(d)∧¬DENUM({b|b∈(ee"d)∧pair(p"(ee"d),b)∈R}))∧SET(e))))) (367)

371 SET(d1) (367)

372 ∃a.(((SET(a)∧SET(c))∧∃k.(a=k∧c=(p"(ee"k))))∧((SET(a)∧SET(c))∧∃d e.%
((a=d∧c=e)∧((NATNUM(d)∧¬DENUM({b|b∈(ee"d)∧pair(p"(ee"d),b)∈R}))∧SET(e))))) (367)

373 SET(c) (367)

374 c=d1≠FALSE (367)

*****ES ↑↑↑↑↑ a;

375 ((SET(a)∧SET(d1))∧∃k.(a=k∧d1=(p"(ee"k))))∧((SET(a)∧SET(d1))∧∃d e.%
((a=d∧d1=e)∧((NATNUM(d)∧¬DENUM({b|b∈(ee"d)∧pair(p"(ee"d),b)∈R}))∧SET(e)))) (375)

*****ES ↑↑↑↑ a1;

376 ((SET(a1)∧SET(c))∧∃k.(a1=k∧c=(p"(ee"k))))∧((SET(a1)∧SET(c))∧∃d e.%
((a1=d∧c=e)∧((NATNUM(d)∧¬DENUM({b|b∈(ee"d)∧pair(p"(ee"d),b)∈R}))∧SET(e)))) (376)

*****ADDFACTS #1#1#1#1#2#2#1#2#1#1#1#1#1#1 ASSUME ↑↑,↑;

Goal #1#1#1#1#2#2#1#2#1#1#1#1#1#1: c1∈B

*****PREPARE;

377 ∃d e.((a=d∧d1=e)∧((NATNUM(d)∧¬DENUM({b|b∈(ee"d)∧pair(p"(ee"d),b)∈R}))∧SET(e))) (375)

378 SET(d1) (367)

379 SET(a) (364)

380 ∃k.(a=k∧d1=(p"(ee"k))) (364)

381 SET(d1) (364)

382 SET(a) (364)

383 ∃d e.((a1=d∧c=e)∧((NATNUM(d)∧¬DENUM({b|b∈(ee"d)∧pair(p"(ee"d),b)∈R}))∧SET(e))) (376)

384 SET(c) (367)

385 SET(a1) (364)

386 ∃k.(a1=k∧c=(p"(ee"k))) (364)

387 SET(c) (364)

388 SET(a1) (364)

*****ES 377 d e;

389 (a=d∧d1=e)∧((NATNUM(d)∧¬DENUM({b|b∈(ee"d)∧pair(p"(ee"d),b)∈R}))∧SET(e)) (389)

*****ES 383 d2 e2;

390 (a1=d2∧c=e2)∧((NATNUM(d2)∧¬DENUM({b|b∈(ee"d2)∧pair(p"(ee"d2),b)∈R}))∧SET(e2)%
) (390)

******es 380 k;

391 a=k∧d1=(p"(ee"k)) (391)

******es 386 j;

392 a1=j∧c=(p"(ee"j)) (392)

******∧E ↑↑ 2;

393 d1=(p"(ee"k)) (391)

******∧E ↑↑ 2;

394 c=(p"(ee"j)) (392)

*****TRY USING REWRITE BY {↑↑,↑ 368};

Goal *1*1*1*1*2*2*1*2*1*1*1*1*1*1*1: pair(p"(ee"j),p"(ee"k))∈B

*****TAUTEQ d2=j ↑↑↑,↑↑↑↑↑;

395 d2=j (390 392)

*****TAUTEQ d=k ↑↑↑↑↑,↑↑↑↑↑↑↑;

396 d=k (389 391)

*****TAUT 389:*2*1*2 389;

397 ¬DENUM({b|b∈(ee"d)∧pair(p"(ee"d),b)∈R}) (389)

*****TAUT 390:*2*1*2 390;

398 ¬DENUM({b|b∈(ee"d2)∧pair(p"(ee"d2),b)∈R}) (390)

*****REWRITE 1↑ BY {1↑↑};

2 substitutions were made

399 ¬DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R}) (375 391)

*****REWRITE 1↑ BY {1↑↑↑};

2 substitutions were made

400 ¬DENUM({b|b∈(ee"j)∧pair(p"(ee"j),b)∈R}) (376 392)

*****ASSUME k=j;

401 k=j (401)

*****REWRITE 391 BY {1};

2 substitutions were made

402 a=j∧d1=(p"(ee"j)) (391 401)

*****TAUTEQ FALSE 369 392 1;

403 FALSE (367 391 392 401)

*****¬I 1 401;

404 ¬(k=j) (367 391 392)

*****REWRITE 1 BY {LESS2};

1 substitutions were made

405 k LT j∨j LT k (367 391 392)

*****TAUTEQ 352:*2∈B 350:352 399 400 1;

406 pair(p"(ee"j),p"(ee"k))∈B (1 13 17 367 375 376 391 392)

*****QED;

407 c1∈B=pair(p"(ee"j),p"(ee"k))∈B (367 391 392)

408 c1∈B (1 13 17 364)

409 (SET(c1)∧∃c d1.((SET(c)∧∃a.(((SET(a)∧SET(c))∧∃k.(a=k∧c=(p"(ee"k))%
))∧((SET(a)∧SET(c))∧∃d e.((a=d∧c=e)∧((NATNUM(d)∧¬DENUM({b|b∈(ee"d)∧pa%
ir(p"(ee"d),b)∈R}))∧SET(e)))))))∧((SET(d1)∧∃a.(((SET(a)∧SET(d1))∧∃k.(a%

=k∧d1=(p"(ee"k)))))∧((SET(a)∧SET(d1))∧∃d e.((a=d∧d1=e)∧((NATNUM(d)∧¬DENUM({b%
|b∈(ee"d)∧pair(p"(ee"d),b)∈R}))∧SET(e))))))∧(¬(c=d1)∧c1=pair(c,d1)))))⊃c1∈B (1 13 17)

410 ∀c1.((SET(c1)∧∃c d1.((SET(c)∧∃a.(((SET(a)∧SET(c))∧∃k.(a=k∧c=(p"(e%
e"k))))∧((SET(a)∧SET(c))∧∃d e.((a=d∧c=e)∧((NATNUM(d)∧¬DENUM({b|b∈(ee"%
d)∧pair(p"(ee"d),b)∈R}))∧SET(e))))))∧((SET(d1)∧∃a.(((SET(a)∧SET(d1))∧%
∃k.(a=k∧d1=(p"(ee"k))))∧((SET(a)∧SET(d1))∧∃d e.((a=d∧d1=e)∧((NATNUM(d%
)∧¬DENUM({b|b∈(ee"d)∧pair(p"(ee"d),b)∈R}))∧SET(e))))))∧(¬(c=d1)∧c1=pa%
ir(c,d1)))))⊃c1∈B) (1 13 17)

411 EDGESET({c|∃a.opair(a,c)∈({b|∃k.b=opair(k,p"(ee"k))}∩CROSS({k|¬DE%
NUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})},V))})⊂B≡∀c1.((SET(c1)∧∃c d1.((S%
ET(c)∧∃a.(((SET(a)∧SET(c))∧∃k.(a=k∧c=(p"(ee"k))))∧((SET(a)∧SET(c))∧∃d%
e.((a=d∧c=e)∧((NATNUM(d)∧¬DENUM({b|b∈(ee"d)∧pair(p"(ee"d),b)∈R}))∧SE%
T(e))))))∧((SET(d1)∧∃a.(((SET(a)∧SET(d1))∧∃k.(a=k∧d1=(p"(ee"k))))∧((S%
ET(a)∧SET(d1))∧∃d e.((a=d∧d1=e)∧((NATNUM(d)∧¬DENUM({b|b∈(ee"d)∧pair(p%
"(ee"d),b)∈R}))∧SET(e))))))∧(¬(c=d1)∧c1=pair(c,d1)))))⊃c1∈B)

412 EDGESET({c|∃a.opair(a,c)∈({b|∃k.b=opair(k,p"(ee"k))}∩CROSS({k|¬DE%
NUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})},V))})⊂B (1 13 17)

413 EDGESET({c|∃a.opair(a,c)∈({b|∃k.b=opair(k,p"(ee"k))}∩CROSS({k|¬DE%
NUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})},V))})⊂B∨EDGESET({c|∃a.opair(a,c%
)∈({b|∃k.b=opair(k,p"(ee"k))}∩CROSS({k|¬DENUM({b|b∈(ee"k)∧pair(p"(ee"%
k),b)∈R})},V))})⊂R (1 13 17)

414 (EDGESET({c|∃a.opair(a,c)∈({b|∃k.b=opair(k,p"(ee"k))}∩CROSS(IF DE%
NUM({k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}) THEN {k|DENUM({b|b∈(e%
e"k)∧pair(p"(ee"k),b)∈R})} ELSE {k|¬DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b%
)∈R})},V))})⊂B∨EDGESET({c|∃a.opair(a,c)∈({b|∃k.b=opair(k,p"(ee"k))}∩C%
ROSS(IF DENUM({k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}) THEN {k|DEN%
UM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})} ELSE {k|¬DENUM({b|b∈(ee"k)∧pair(%
p"(ee"k),b)∈R})},V))})⊂R)≡(EDGESET({c|∃a.opair(a,c)∈({b|∃k.b=opair(k,%
p"(ee"k))}∩CROSS({k|¬DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})},V))})⊂B∨%
EDGESET({c|∃a.opair(a,c)∈({b|∃k.b=opair(k,p"(ee"k))}∩CROSS({k|¬DENUM(%
{b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})},V))})⊂R) (299)

415 EDGESET({c|∃a.opair(a,c)∈({b|∃k.b=opair(k,p"(ee"k))}∩CROSS(IF DEN%
UM({k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}) THEN {k|DENUM({b|b∈(ee%
"k)∧pair(p"(ee"k),b)∈R})} ELSE {k|¬DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)%
∈R})},V))})⊂B∨EDGESET({c|∃a.opair(a,c)∈({b|∃k.b=opair(k,p"(ee"k))}∩CR%
OSS(IF DENUM({k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}) THEN {k|DENU%
M({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})} ELSE {k|¬DENUM({b|b∈(ee"k)∧pair(p%
"(ee"k),b)∈R})},V))})⊂R (1 13 17 299)

416 ¬DENUM({k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})})⊃(EDGESET({c|∃a%
.opair(a,c)∈({b|∃k.b=opair(k,p"(ee"k))}∩CROSS(IF DENUM({k|DENUM({b|b∈%
(ee"k)∧pair(p"(ee"k),b)∈R})}) THEN {k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k)%
,b)∈R})} ELSE {k|¬DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})},V))})⊂B∨EDG%
ESET({c|∃a.opair(a,c)∈({b|∃k.b=opair(k,p"(ee"k))}∩CROSS(IF DENUM({k|D%
ENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}) THEN {k|DENUM({b|b∈(ee"k)∧pair%
(p"(ee"k),b)∈R})} ELSE {k|¬DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})},V))})⊂R) (1 13 17)

417 EDGESET({c|∃a.opair(a,c)∈({b|∃k.b=opair(k,p"(ee"k))}∩CROSS(IF DEN%
UM({k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}) THEN {k|DENUM({b|b∈(ee%

"k)∧pair(p"(ee"k),b)∈R)}} ELSE {k|¬DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)%
∈R)}),V))}}⊂B∨EDGESET({c|∃a.opair(a,c)∈({b|∃k.b=opair(k,p"(ee"k))}∩CR%
OSS(IF DENUM({k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}) THEN {k|DENU%
M({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})} ELSE {k|¬DENUM({b|b∈(ee"k)∧pair(p%
"(ee"k),b)∈R})},V))}}⊂R (1 13 17)

418 (EDGESET(RNG{b|∃k.b=opair(k,p"(ee"k))} | IF DENUM({k|DENUM({b|b∈%
(ee"k)∧pair(p"(ee"k),b)∈R})}) THEN {k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k)%
,b)∈R})} ELSE {k|¬DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}))⊂B∨EDGESET%
(RNG({b|∃k.b=opair(k,p"(ee"k))} | IF DENUM({k|DENUM({b|b∈(ee"k)∧pair(%
p"(ee"k),b)∈R})}) THEN {k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})} ELS%
E {k|¬DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}))⊂R)≡(EDGESET({c|∃a.opa%
ir(a,c)∈({b|∃k.b=opair(k,p"(ee"k))}∩CROSS(IF DENUM({k|DENUM({b|b∈(ee"%
k)∧pair(p"(ee"k),b)∈R})}) THEN {k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈%
R})} ELSE {k|¬DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})},V))})⊂B∨EDGESET%
({c|∃a.opair(a,c)∈({b|∃k.b=opair(k,p"(ee"k))}∩CROSS(IF DENUM({k|DENUM%
({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}) THEN {k|DENUM({b|b∈(ee"k)∧pair(p"%
(ee"k),b)∈R})} ELSE {k|¬DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})},V))})⊂R)

419 EDGESET(RNG({b|∃k.b=opair(k,p"(ee"k))} | IF DENUM({k|DENUM({b|b∈(%
ee"k)∧pair(p"(ee"k),b)∈R})}) THEN {k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),%
b)∈R})} ELSE {k|¬DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}))⊂B∨EDGESET(%
RNG({b|∃k.b=opair(k,p"(ee"k))} | IF DENUM({k|DENUM({b|b∈(ee"k)∧pair(p%
"(ee"k),b)∈R})}) THEN {k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})} ELSE%
{k|¬DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}))⊂R (1 13 17)

420 DENUM(RNG({b|∃k.b=opair(k,p"(ee"k))} | IF DENUM({k|DENUM({b|b∈(ee%
"k)∧pair(p"(ee"k),b)∈R})}) THEN {k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)%
∈R})} ELSE {k|¬DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}))∧(EDGESET(RNG%
({b|∃k.b=opair(k,p"(ee"k))} | IF DENUM({k|DENUM({b|b∈(ee"k)∧pair(p"(e%
e"k),b)∈R})}) THEN {k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})} ELSE {k%
|¬DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}))⊂B∨EDGESET(RNG({b|∃k.b=opa%
ir(k,p"(ee"k))} | IF DENUM({k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}%
) THEN {k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})} ELSE {k|¬DENUM({b|b%
∈(ee"k)∧pair(p"(ee"k),b)∈R})}))⊂R) (1 13 17)

421 RNG({b|∃k.b=opair(k,p"(ee"k))} | IF DENUM({k|DENUM({b|b∈(ee"k)∧pa%
ir(p"(ee"k),b)∈R})}) THEN {k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})} %
ELSE {k|¬DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})})⊂G∧(DENUM(RNG({b|∃k.%
b=opair(k,p"(ee"k))} | IF DENUM({k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)%
∈R})}) THEN {k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})} ELSE {k|¬DENUM%
({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}))∧(EDGESET(RNG({b|∃k.b=opair(k,p"(%
ee"k))} | IF DENUM({k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}) THEN {%
k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})} ELSE {k|¬DENUM({b|b∈(ee"k)∧%
pair(p"(ee"k),b)∈R})}))⊂B∨EDGESET(RNG({b|∃k.b=opair(k,p"(ee"k))} | IF%
DENUM({k|DENUM({b|b∈(ee"k)∧pair(p"(ee"k),b)∈R})}) THEN {k|DENUM({b|b%
∈(ee"k)∧pair(p"(ee"k),b)∈R})} ELSE {k|¬DENUM({b|b∈(ee"k)∧pair(p"(ee"k%
),b)∈R})}))⊂R)) (1 13 17)

422 ∃a.(a⊂G∧(DENUM(a)∧(EDGESET(a)⊂B∨EDGESET(a)⊂R))) (1)

423 (DENUM(G)∧(EDGESET(G)=(R∪B)∧(R∩B)=λ))⊃∃a.(a⊂G∧(DENUM(a)∧(EDGESET(%
a)⊂B∨EDGESET(a)⊂R)))

424 ∀G R B.((DENUM(G)∧(EDGESET(G)=(R∪B)∧(R∩B)=λ))⊃∃a.(a⊂G∧(DENUM(a)∧(%

EDGESET(a)⊂B∨EDGESET(a)⊂R))))

8.5. Statistics of the proof.

For the proof shown in the last section, the user typed 309 commands. Of these, 199 were *forward* FOL commands. 110 were GOAL commands properly, including 14 commands for goal creation, 6 for addition of *facts*, and 72 calls to TRY.

The complete statistics are shown next.

Goal commands:

GOAL	14
ADDFACTS	6
QED	6
PREPARE	14
TRY	72
TOTAL	110

Detail of TRY:

∀I	3
⊃I	2
∧I	3
REWRITE	21
MONADIC	9
∃I	3
ELIMINATION	12
LOGIC	3
TAUT	4
TAUTEQ	3
IMPLICATION	2
INDUCTION	2
UNIFY	1
EQUNIFY	1
IFCASES	1
∨I	2

Summary of FOL commands:

LABEL	21
REWRITE	35
MONADIC	10
RESOLVE	4
∀E	46
∀I	8
∃E	23
∃I	1
TAUT	20
TAUTEQ	12
SIMPLIFY	2
EVAL	6
∧E	3
SUBSTR	3
DED	2
ASSUME	2
¬I	2
TOTAL	199

8.6. Conclusion.

Summing up the statistics just shown with the 44 commands used in the proof of the auxiliary lemmas, we can conclude that the old proof required roughly twice as many user commands as this one.

This is not as great a gain as we had hoped for, in terms of just the number of commands. However, there are other gains: the proof of Ramsey's theorem is very complex, and the ability to work on several goal trees seems to make it much easier to construct the proof. At least this is true in my own experience.

Ramsey's does not seem to be the kind of theorem where the reduction in the number of commands is largest. In the auxiliary theorems proved earlier, the reduction was by a factor of four. Those theorems are of medium size: their FOL proofs were between 10 and 50 lines each. It probably is for small and medium size theorems where the greatest reduction in the number of user commands can be achieved by GOAL. At the same time, it is probably for the more complex theorems like Ramsey's that the advantage of GOAL as an aid for structured, top down proof construction is more likely to be felt.

9. REFERENCES.

Abrahams 1963
Abrahams, P. W. , *Machine Verification of Mathematical Proof.* Doctoral Dissertation, M.I.T., Cambridge, Mass., May 1963.

Allen and Luckham 1970
Allen, John and Luckham, David, *An Interactive Theorem-Proving Program*, Machine Intelligence, 5, 1970, 321-336.

Bledsoe 1971
Bledsoe, W. W., *Splitting and Reduction Heuristics in Automatic Theorem Proving.* Artificial Intelligence, 2, 1971, 55-77.

Bledsoe and Bruell 1974
Bledsoe, W. W. and Bruell, Peter, *A Man-Machine Theorem-Proving System.* Artificial Intelligence, 5, 1974, 51-72.

Bledsoe and Gilbert 1967
Bledsoe, W. W. and Gilbert, E. J., *Automatic Theorem Proof-Checking In Set Theory: a preliminary report.* Sandia Corp. Rept. SC-RR-67-525, July 1967.

Bledsoe, Boyer and Henneman 1972
Bledsoe, W. W., Boyer, R.S. and Henneman, W.H., *Computer Proofs of Limit Theorems.* Artificial Intelligence, 3, 1972, 27-60.

Brown 1977a
Brown, F. M, *A Theorem Prover for Elementary Set Theory.* Proc. 5th International Joint Conference on Artificial Intelligence, August 1977, 534-540.

Brown 1977b
Brown, F. Malloy, *Doing Arithmetic Without Diagrams.* Artificial Intelligence, 8, 1977, 175-200.

Brown 1978
Brown, F. M, *Towards the Automation of Set Theory and Its Logic.* Submitted to Artificial Intelligence.

Bundy 1973
Bundy, A., *Doing Arithmetic With Diagrams.* Proc. Third Int. Joint Conf. on Artificial Intelligence, Stanford, Ca., August 1973, 56-65.

Cartwright 1976
Cartwright, Robert, Practical Formal Semantic Definition and Verification Systems. Ph.D. Thesis, Stanford University, Computer Science Department, AIM-296, December 1976.

Cartwright and McCarthy 1979
Cartwright, Robert and McCarthy, John, **Recursive Programs as Functions in a First Order Theory.** Stanford University, Computer Science Department, AIM-324, 1979.

De Bruijn 1970
De Bruijn, N. G., *The Mathematical Language AUTOMATH, Its usage, and some of its extensions*, in **Lecture Notes in Mathematics, Vol. 125,** Springer Velag, 1970, p. 29-61.

De Bruijn 1971
De Bruijn, N. G., *AUTOMATH, a Language for Mathemmatics.* Notes of a series of lectures in the Séminaire de Mathématiques Supérieures, Université de Montreal, 1971.

De Bruijn 1974
De Bruijn, N. G., *The AUTOMATH Mathematics Checking Project.* Report, Department of Mathematics, Technological University, Eindhoven, The Netherlands, 1974.

Ernst 1971
Ernst, G., *The Utility of Independent Subgoals in Theorem Proving.* **Information and Control, 18** (3), 1971.

Gentzen 1935
Gentzen, G., *Untersuchungen ueber das logische Schliessen*, **Mathematische Zeitschrift, 39,** 1934-5.

Goldstein 1973
Goldstein, I., *Elementary Geometry Theorem Proving.* A.I. Memo 280, M.I.T., Canbridge, Mass., April 1973.

Gordon, Milner and Wadsworth 1977
Gordon, M., Milner, R., and Wadsworth, C., Edinburgh LCF, Department of Computer Science Internal Report CSR-11-77, University of Edinburgh, 1977.

Gordon, Milner, Morris, Newey and Wadsworth 1978
Gordon, M., Milner, R., Morris, L., Newey, M. and Wadsworth, C., *A Metalanguage for Interactive Proof in LCF*, Fifth ACM Conference on Principles of Programming Languages, Tucson, Arizona, 1978.

Jutting 1977
Jutting, L. S. van Benthem, **Checking Landau's GRUNDLAGEN in the AUTOMATH system,** Thesis, Technische Hogeschool, Eindhoven, 1977.

Kelley 1955
Kelley, John L., **General Topology,** Van Nostrand, Princeton, N.J., 1955.

Lee 1967
Lee, Richard Char-Tung, **A Completeness Theorem and a Computer Program for**

Finding Theorems Derivable from Given Axioms, doctoral dissertation, Department of Electrical Engineering and Computer Science, University of California, Berkeley, 1967.

Luckham 1967
Luckham, D., *The Resolution Principle in Theorem-Proving*, Machine Intelligence, 1, 1967, 47-61.

McCarthy 1962
McCarthy, John, *Computer Programs for Checking Mathematical Proofs*. Amer. Math. Soc. Proc. Symposia in Pure Math., Vol. 5, 1962.

McCarthy 1963
McCarthy, John, *A Basis for a Mathematical Theory of Computation*, in P. Biaffort and D. Hershberg (eds.), Computer Programming and Formal Systems, North-Holland, Amsterdam, 1963.

McCarthy 1965
McCarthy, John, A Proof-Checker for Predicate Calculus. Stanford University, Computer Science Department, AIM-27, March 1965.

McCarthy 1966
McCarthy, John, *A Formal Description of a Subset of Algol*, in T. Steele (ed.), Formal Language Description Languages for Computer Programming, North-Holland, Amsterdam, 1966.

McCarthy 1977
McCarthy, John, Representation of Recursive Programs in First Order Logic. Unpublished draft of a technical report, February 1977.

McCarthy 1978a
McCarthy, John, An Interesting LISP Function. Unpublished manuscript.

McCarthy 1978b
McCarthy, John, An Example of Case Analysis Involving Inequalities in FOL. Unpublished manuscript.

McCarthy 1979
McCarthy, John, First Order Theories of Individual Concepts and Propositions. Forthcoming.

McCarthy and Painter 1967
McCarthy, John and Painter, James, *Correctness of a Compiler for Arithmetic Expressions*, Amer. Math. Soc., Proc. Symposia in Applied Math., Math. Aspects of Computer Science, New York, 1967.

McCarthy, Sato, Hayashi and Igarashi 1978
McCarthy, J., Sato, M., Hayashi, T. and Igarashi, S., *On the Model Theory of Knowledge*. Stanford University, Computer Science Department, AIM-312, April 1978.

Mendelson 1964
Mendelson, Elliott, Introduction to Mathematical Logic Van Nostrand Reinhold Co., N.Y., 1964.

Milner 1972a
Milner, Robin, *Implementation and Application of Scott's Logic for Computable Functions.* Proc. ACM Conf. on Proving Assertions about Programs. New Mexico State University, Las Cruces, New Mexico, 1972, p. 1-6.

Milner 1972b
Milner, Robin, Logic for Computable Functions: Description of a Machine Implementation. Stanford University, Computer Science Department, AIM-169, May 1972.

Milner 1973
Milner, Robin, Models of LCF. Stanford University, Computer Science Department, AIM-186, January 1973.

Milner and Weyhrauch 1972a
Milner, Robin and Weyhrauch, Richard, *Program Semantics and Correctness in a Mechanized Logic*, Proc. 1st. USA-Japan Computer Con., Tokyo, 1972.

Milner and Weyhrauch 1972b
Milner, Robin and Weyhrauch, Richard, *Proving Compiler Correctness in a Mechanized Logic*, Machine Intelligence, 7, Edinburgh University Press, 1972.

Morales 1973
Morales, Jorge, *Interactive Theorem Proving.* Proc. ACM National Conference, August 1973.

Nevins 1974
Nevins, Arthur J., *A Human Oriented Logic for Automatic Theorem-Proving.* Journal of the ACM, 21, Nr. 4, October 1974, 505-621.

Nevins 1975a
Nevins, Arthur J., *Plane Geometry Theorem Proving Using Forward Chaining.* Artificial Intelligence, 6, 1975, 1-23.

Nevins 1975b
Nevins, Arthur J., *A Relaxation Approach to Splitting in an Automatic Theorem Prover.* Artificial Intelligence, 6, 1975, 25-39.

Nilsson 1971
Nilsson, N. J., Problem-Solving Methods in Artificial Intelligence, McGraw-Hill, New York, 1971.

Pastre 1978
Pastre, D., *Automatic Theorem Proving In Set Theory.* Artificial Intelligence, 10, 1978, 1-27.

Prawitz 1965
Prawitz, D., **Natural Deduction - a Proof-Theoretical Study**, Almqvist & Wiksell, Stockholm, 1965.

Robinson 1965
Robinson, J.A., *A Machine Oriented Logic Based on the Resolution Principle*. **Journal of the ACM 12**, January 1965, 23-41.

Scott 1969
Scott, Dana, **Models for the λ-Calculus**. Unpublished Manuscript, 1969.

Scott and Strachey 1972
Scott, D. S. and Strachey, C., *Towards a Mathematical Semantics for Computer Languages*, Proc. Symposium on Computers and Automata, **Microwave Research** Institute Symposia Series, Vol. 21, Polytechnic Institute of Brooklyn, 1972.

Shoenfield 1967
Shoenfield, J. R., **Mathematical Logic**, Addison-Wesley Publ. Co., 1967.

Slagle 1971
Slagle, James R., **Artificial Intelligence: The Heuristic Programming Approach**, McGraw-Hill, New York, 1971.

Slagle 1976
Slagle, J. R., *Theorem Proving*. In Ralston, A. (Ed.), **Encyclopedia of Computer Science**, Petrocelly/Charter, N.Y. 1976.

Smith and Blaine 1976
Smith, Robert L. and Blaine, Lee H., *A Generalized System for University Mathematics Instruction*, in Colman, R. and Lorton, P., (Eds.), **Computer Science and Education**, Proc. ACM SIGCSE-SIGCUE Joint Symposium, 1976.

Suppes 1975
Suppes, Patrick, *Impact of Computers on Curriculum in the Schools and Universities*, In Lecarme, O. and Lewis, R., (Eds.), **Computers in Education**, IFIP, North-Holland, 1975.

Wagner 1977
Wagner, Todd, **Hardware Verification**. Ph.D. Thesis, Stanford University, Computer Science Department, AIM-304, September 1977.

Weyhrauch 1976
Weyhrauch, Richard, *Practical Program Verification: Are We Close?* Meeting on 20 Years of Computer Science, University of Pisa, Italy, June 1976.

Weyhrauch 1977
Weyhrauch, Richard W., **A Users Manual for FOL**. Stanford University, Computer Science Department, AIM-235.1, July 1977.

Weyhrauch 1978a

Weyhrauch, Richard W., *Prolegomena to a theory of mechanized formal reasoning*, submitted to **Artificial Intelligence**, also available as: Artificial Intelligence Laboratory Memo AIM-315, Stanford University, October 1978.

Weyhrauch et al. 1979

Weyhrauch, Richard (ed.), **Proofs Using FOL.** Forthcoming Stanford Artificial Intelligence Memo.

10. INDEX.