Copyright
by
Benjamin Price Shults
1997

DISCOVERIES AND EXPERIMENTS IN THE AUTOMATION OF MATHEMATICAL REASONING

by

BENJAMIN PRICE SHULTS, B.A.

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT AUSTIN

December 1997

DISCOVERIES AND EXPERIMENTS IN THE AUTOMATION OF MATHEMATICAL REASONING

APPROVED BY
DISSERTATION COMMITTEE:

Supervisor: _____

Supervisor: _____

_____

_____

_____

For Alice

Acknowledgements

Even before starting graduate school, I had read about the ATP program at The University of Texas at Austin and saved the information with the thought that someday I might play with it. I had sometimes commented to my classmates that proving certain kinds of theorems was so monotonous that a computer should be able to do it. When Norm McCain gave a guest lecture on resolution and automated theorem proving in Vladimir Lifschitz' logic course during my first year of graduate school, I was inspired but not immediately convinced to change from my plan to study topology. If it hadn't been for Norm's lecture, I may never have considered it seriously.

Later, when I spoke to Woody Bledsoe about the possibility of working with him, his encouragement was all I needed. I owe him thanks for encouraging me when I was about to give up and for giving me a guiding vision for this work. Woody Bledsoe was my supervisor during the bulk of this work until his death in October of 1995.

After Woody's death, Larry Hines supervised my work. I am very grateful to him for his careful reading and thoughts about my work. He was very good at communicating his intelligent recommendations.

At the same time, Robert Boyer became my co-supervisor pro forma. I thank him for setting high standards and making excellent recommendations for changes in my dissertation. Dr. Boyer also has a knowledge of the folklore of the field and I appreciated his passing on stories from the past.

Working with Benjamin Kuipers on a somewhat unrelated project in artificial intelligence helped to shape me as a writer and a professional researcher. I also thank Benjamin Kuipers for taking me into his group, giving me the opportunity to work with him and for being bold enough to give advice. His advice was all excellent and badly needed.

I am very grateful to all of my committee members. I thank Robert Causey for going far beyond the call of a committee member to get involved in my work. I thank Vladimir Lifschitz for inspiration and encouragement. I thank Michael Starbird for encouragement and friendliness throughout my graduate career.

Many visitors to UT and researcher at places I visited helped, inspired and encouraged me with their own research and with their kind words to me.

I am grateful to my family for their advice and support. My wife, Alice, loved me, read drafts of almost every paper I wrote, worked full-time and kept me fed. My father and older brother, who have both been through the process of getting a PhD, gave excellent advice and support. The rest of my family was available any time I needed them and being with them is always a joy.

With the exception of the period during which I was working with Ben Kuipers and the summers when I was working on research, I was also teaching and tutoring mathematics. I am grateful to the administrators of the Department of Mathematics for offering me this opportunity and support.

I am very thankful to Mark Catalano for running around UT getting signatures for me after I had moved to Ohio. I am also thankful to the faculty of the math department at Kenyon College for letting me take the time to defend my dissertation just before classes began.

Finally, I thank Jesus for doing more for me than I could ever ask or imagine.

DISCOVERIES AND EXPERIMENTS IN THE AUTOMATION OF MATHEMATICAL REASONING

Publication No. _____

Benjamin Price Shults, Ph.D.
The University of Texas at Austin, 1997

Supervisors: Larry Hines
Robert Boyer

The problem of automatically proving that a statement follows logically from a base of knowledge is recurrent in artificial intelligence and automated theorem proving research. It is a problem that mathematicians must handle very well in order to prove theorems in mathematics. One of the difficulties of proving a statement is choosing when and how to use the knowledge one has about the topic. Since this is difficult for mathematicians, we can expect that it will be difficult for machines.

In this dissertation, the author describes a framework for approaching this problem and illustrates its success on certain kinds of problems. A simple and tractable calculus is also described for handling parts of set theory and equality. These devices allow for more complicated mathematical expressions and theories to be handled more easily.

The rules presented here for storing and applying knowledge were designed with the aim of facilitating human interface. Knowledge is stored in a format that is much more cognitively sensible than the standard clausal form. Knowledge is applied during the search for a proof in a way that is very similar to the way humans explain the application of knowledge in proofs.

The computer program, IPR, uses the advantages of the framework not only to find proofs of statements in theories under difficult circumstances but also to provide features that

make it easy to be used by someone who knows nothing about automated methods but who wants to get directly into mathematical reasoning. The method is very easy to implement and facilitates excellent interactive capabilities including help for the user in understanding an unfinished proof.

Table of Contents

Acknowledgements **v**

Abstract **vii**

List of Figures **xii**

Chapter 1. Introduction **1**

1.1 Reasoning in Mathematics 3

1.1.1 Reasoning with Knowledge 3

1.1.2 Higher-Order Reasoning 5

1.2 Interface Issues 6

1.2.1 Output 6

1.2.2 Input 7

1.2.3 Interaction 7

1.3 A Survey of the Automation of Reasoning 8

1.4 An Overview of the Dissertation 16

Chapter 2. Formal Reasoning **18**

2.1 Language for a Theory 19

2.1.1 Language Issues 20

2.1.2 First-Order Logic 21

2.2 Reasoning in a Theory 25

2.2.1 Tableaux 27

2.2.2 Sequent Calculus 33

2.3 Knowledge in a Theory 36

2.3.1 Clausal Form 38

2.3.2 Rules for Transformation into Non-Clausal Form 43

2.3.3 Soundness and Completeness 45

Chapter 3. Application of Knowledge **48**
3.1 Informal Description . . . 49
3.2 Technical Details . . . 56
3.2.1 Unification Strategies . . . 57
3.2.2 The ε-Rule . . . 59
3.2.3 Soundness and Completeness . . . 63
3.3 Selection of Knowledge . . . 67

Chapter 4. Extensions to First-Order Logic **73**
4.1 Equality . . . 73
4.1.1 Brown's Rule . . . 74
4.1.2 Congruence Closure . . . 74
4.1.3 Miscellaneous Equality Rules . . . 75
4.2 Comprehension and Description in Tableaux . . . 78
4.2.1 The Comprehension Schema in Tableaux . . . 80
4.2.2 Description in Tableaux . . . 83
4.2.3 Axioms of Set Theory . . . 83
4.2.4 Handling Equality in Set Theory . . . 85
4.2.5 Control Strategies . . . 88
4.2.6 Examples . . . 89
4.2.7 Remarks . . . 98

Chapter 5. IPR: Control and Interface **101**
5.1 Control Issues . . . 102
5.1.1 Breadth-First Unification . . . 102
5.1.2 High-level Control . . . 108
5.2 The User Interface . . . 114
5.2.1 The Input Language . . . 116
5.2.2 Input . . . 117
5.2.3 Interaction . . . 120
5.2.4 Output . . . 125

Chapter 6. Examples **127**

Chapter 7. Related Work **145**
7.1 Reasoning 146
7.2 Interface 149

Chapter 8. Conclusion **151**
8.1 Summary 151
8.2 A Brief History of IPR 153
8.3 The Theorem $n + 1$ Challenge 155
8.4 Imitating Human Techniques. 159
8.5 Synergistic Effect 160
8.6 Future Work 161

Bibliography **167**

Vita **180**

List of Figures

2.1 The signed tableau (A) and ordinary tableau (B) for Example 2.2. 30
2.2 The signed tableau for Example 2.3. The symbols f_1 and f_2 are Skolem constants. 31
2.3 The bottom of the instantiated signed tableau for Example 2.3. The symbols f_1 and f_2 are Skolem constants. 32

3.1 The proof for Example 3.1. 50
3.2 The tableau for Example 3.3. The symbols X_0, A_0 and a_0 are Skolem constants. 62
3.3 The tableaux for Example 3.4 69

4.1 The knowledge base and proof excerpt for Example 4.1. 76
4.2 The tableaux for Example 4.1. The symbols a, b, S and T are constants. 77
4.3 Some examples of theorems involving ι-terms. 84
4.4 The proof of $a = c$ in Example 4.4. The symbols a, b, c and d are Skolem constants. 92
4.5 The proof of $b = d$ in Example 4.4. 94
4.6 Part of the tableau for Example 4.4. 95
4.7 The tableau for Example 4.5. The symbols X, Y, a_0 and b_0 are Skolem constants. 97

5.1 The tableau for Example 5.1 after one step. 110
5.2 The tableau for Example 5.1 after two steps. 111
5.3 The tableau for Example 5.1. 111
5.4 The input for Example 5.3. 119
5.5 The knowledge for Example 5.3 as IPR displays it. 119
5.6 The knowledge for Example 5.4 as IPR displays it. 121
5.7 The interaction for Example 5.4. 122
5.8 The condensed proof tree for Example 5.4. 124
5.9 The automatically generated English proof for Example 5.5. 125

6.1 The first part of the knowledge base formed for Example 6.1. 130
6.2 The second part of the knowledge base formed for Example 6.1. 131
6.3 The top of the tableau for Example 6.1. 132

6.4 The middle of the tableau for Example 6.1. 133
6.5 The bottom of the tableau for Example 6.1. 135
6.6 Part of the interaction for Example 6.1. 137
6.7 The tableau for Example 6.2. 139
6.8 IPR's proof of Example 6.2. 139
6.9 The knowledge base sufficient for Example 6.3. 141
6.10 The proof tree for Example 6.3. 142
6.11 The input for Example 6.5. 144

8.1 The knowledge for the challenge. 166

Chapter 1

Introduction

> The following question now arises as a fundamental problem: Is it possible to determine whether or not a given statement pertaining to a field of knowledge is a consequence of the axioms? ... [We] are justified in calling it the main problem of mathematical logic.
>
> —David Hilbert [67]

While automatic methods have had great success in many areas of mathematical reasoning, large classes of problems remain out of reach for machines. In particular, the process of establishing the truth of a statement given a large base of knowledge in the theory is very difficult to automate. Typically, when a computer program is given a large base of knowledge and asked to prove that some statement is a consequence of the knowledge, the program fails due to the size and complexity of the search space. The author presents a framework for storing knowledge and accessing it in the process of finding a proof of a new statement that is a consequence of the knowledge. The framework includes a calculus for proving theorems and strategies for controlling the application of rules in the calculus.

The primary high-level motivation behind this work is to have a computer program that finds the proofs of (i.e., establishes the truth of) theorems in mathematical theories automatically. The second high-level motivation behind the development of this framework is that the rules be sensible to non-experts so that the program implementing the rules will be usable by a person who may be an expert in mathematics but is not an expert at automatic techniques. These two basic considerations influenced many of the choices made in the development of the calculus. A program using the calculus presented in this dissertation can

easily interact with humans and give useful feedback including explanations of proofs and proof attempts.

The calculus and strategies presented in this dissertation have been demonstrated to be successful by the fact that a program (IPR) using the framework has proved traditionally difficult problems with ease. The framework for storing and using knowledge used in IPR give it an ability to prove theorems in advanced mathematics. The IPR system takes advantage of the nature of the framework to provide a nice user interface.

We first sum up the novelties in this approach. IPR uses a novel non-clausal form for storing knowledge in a theory (Section 2.3.) The knowledge is applied using the author's ε-rule (Section 3.2.) Since the application of knowledge using the ε-rule resembles the way a mathematician might explain the application of knowledge, it is easy to invent strategies for selecting knowledge that agree with common sense. The strategies introduced here try to keep the search space small while selecting the most closely related knowledge (Section 3.3.) IPR also uses some novel methods to control the search for a proof (Section 5.1.) In order to allow easy expression of common mathematical concepts, IPR extends first-order logic to include the classifier, $\{x : P\}$. A new calculus for handling the classifier within a tableau calculus is described (Section 4.2.) The calculi were developed with human understanding in mind. Therefore, it has been relatively easy to implement an unusually informative interface to the prover (Section 5.2.) IPR also explains the proofs it finds completely in English (Section 5.2.4.) All of the novelties mentioned so far were developed by 1994 and published in at least a primitive form in early technical reports [106, 107]. Since that time, most of the work has gone into improving the implementation to the point that it could prove theorems that are difficult for other systems. Some examples of theorems proved are detailed in Chapter 6.

We begin with a discussion of some of the issues related to the main motivations for this work. We will then give a quick overview of the history of the automation of reasoning (Section 1.3.) In this overview, many of the basic terms that will be used throughout the dissertation are introduced.

We divide this discussion into two areas: the type of reasoning that we want the

computer to do (Section 1.1) and the kind of interface we want it to have (Section 1.2.)

1.1 Reasoning in Mathematics

Reasoning in mathematics requires a tremendous variety of types of reasoning. In this project, we only attack a few of these types of reasoning. There is a discussion of some types of mathematical reasoning that remain out of reach in Section 8.3, and types of reasoning implemented in other systems in Section 1.3.

The types of reasoning that have been attempted by the IPR system in novel ways can be divided into two categories: reasoning with knowledge and higher-order (set theoretic) reasoning.

1.1.1 Reasoning with Knowledge

When trying to prove a statement in mathematics, a mathematician knows many thousands of definitions, theorems and axioms. Somehow, the mathematician is often able to select from that base of knowledge just the information that is needed for the proof. Humans are by no means perfect at this task: they frequently waste time applying theorems that are not needed in the proof that is finally found. Sometimes a person will spend years trying to prove some statement without success. (Fermat's last theorem comes to mind.) Since reasoning with a lot of knowledge is difficult for people, we can expect that it will be difficult for computers.

Nevertheless, a computer with the ability to reason well with knowledge would be very useful. Mathematicians, physicists, engineers, economists and others could use it to verify that their hypotheses are in agreement with the knowledge in their field or with certain other assumptions. Related technology could be used in artificial intelligence programs that are used to do common-sense reasoning in particular.

In talking about proving theorems, Solow makes an analogy with solving a maze [114]. One can work forward or backward: one can concentrate on the statement being proved and try to decide what would be sufficient, or one can concentrate on the assumptions and try to draw conclusions from them until the goal is reached. As Solow mentions, we usually work

forward and backward alternatively.

> It might be necessary to alternate several times between the forward and backward processes before you succeed, for there are likely to be several false starts and blind alleys.
>
> —Daniel Solow [114]

Consider how complicated this maze is. Suppose that at each step in the proof (maze) there are only 100 different ways of applying the theorems we know. This means that in each step in the maze, there are 100 different possible directions we could go. So there are 10,000,000,000 different places we could be after making only five decisions in the maze. In addition, this maze has many infinitely long corridors that *never* cycle. This is a *very* complex maze.

Now there are several different ways computers can be used to solve mazes. One is called the depth-first method. In this method, the computer follows a single path until it comes to a dead end. Then it backs up to its last decision and chooses a different direction. This repeats until a solution is found. This method may not work if our maze has paths that are infinitely long. The computer may follow a path that goes on forever without a dead end.

A second method computers use to solve a maze is called the breadth-first method. In this method, the computer tries all corridors at once. It takes one step down each corridor, then one more step down each corridor until it finds the exit. If there is a finite path through the maze, this method will find it. But how long will it take? After only five steps, the computer will have to have considered 10,101,010,101 places in the maze. The number of places considered grows exponentially. An additional problem for this method is that in some logics, there are *infinitely many* different directions one can choose from any given point in the maze. In these logics, the breadth-first method cannot succeed at all.

A third method is to apply one of the above methods or a combination of them with some kind of *strategy* for choosing the direction to go or not to go. At least on the conscious level, this is what humans do. The IPR program uses this third approach. Section 3.3 mentions some of the strategies it uses.

Apart from the sheer immensity of the search space, there are other factors that make using a large knowledge base difficult. One reason relates to the fact that so much depends on the choice of knowledge to apply and how to apply it. If a wrong decision is made then the rest of the proof attempt is pointed in a direction that may not lead to a proof. For instance, if a variable is instantiated in order to apply a known statement, then you have to wonder if that variable instantiation is what is correct for the rest of the proof. When the application of a theorem introduces a new predicate then you must wonder if that predicate should become your focus and perhaps further knowledge selection should trigger on it. And so once a wrong decision is made, the proof becomes much harder to finish.

Briefly, the logical framework presented here is a combination of some novel methods for reasoning that allow for a sensible strategy to have success in some difficult problems. The first novelty is the format in which knowledge is stored (Section 2.3.2.) Knowledge is stored in a sensible and useful non-clausal format that facilitates automatic use and human understanding [110]. This is useful in an automated theorem proving context because it allows humans to interact if necessary and to understand the output proof. The second novelty is the first-order calculus used for applying knowledge while constructing tableaux (Section 3.2.) Knowledge is applied in a way that resembles the way mathematicians explain the use of theorems. The combination of these two novelties along with the novel control methods discussed in Section 5.1 make possible the success of certain strategies for automatically selecting knowledge for use in the process of finding a proof (Section 3.3.)

The reason the strategy is important is that the maze is very complex. The IPR program, like the mathematician, tries to narrow down the number of paths that are likely to lead to success and tries to go in the direction that leads most directly to the proof. This is accomplished by means of the simple "$n + m$" strategy and other strategies (Section 3.3.)

1.1.2 Higher-Order Reasoning

While first-order logic (Section 2.1.2) is capable, in theory, of expressing anything in ordinary mathematics, higher-order logic allows for more convenient and brief expressions to be used. The IPR system utilizes a novel method for proving some theorems in higher-order logic.

Higher-order logic is very useful in mathematics because it allows more complex sentences to be more easily expressed and proved. However, when the language is extended in this way, reasoning typically becomes much more complex.

Several methods have been developed for handling higher-order reasoning. Some have developed provers for full higher-order logic [2, 4]. Others have implemented Gödel's axioms in order to keep set theory completely within first-order logic [102]. Others have despaired of handling the problem automatically and resorted to proof checkers [96]. The method used by Frank Brown [39] turns out to be rather closely related to the method described in this dissertation.

The method used by IPR is essentially the implementation of the axiom schema of comprehension by means of some new tableau branch closure rules and new tableau branch expansion rules. This will be explained in detail in Section 4.2. This very simple and tractable mechanism apparently encompasses a great deal of common higher-order reasoning in a relatively expressive language.

The strength of this method is also illustrated by showing that it proves very easily several examples that have been considered difficult for other automatic methods.

1.2 Interface Issues

One of the barriers that keeps mathematicians from using automated theorem proving software is the interface. In order to use most systems, the user really must be an expert at the underlying proof calculus. One of the primary goals in the development of the calculus used by IPR is that it support an intuitive user interface and allow the user to understand what is happening without learning a new logic.

1.2.1 Output

In the case of many automatic theorem provers, the output is limited to one of the following phrases: "the formula is a theorem," "the formula is not a theorem" or "unable to determine." This is not very useful information. The mathematician wants to know why it is a theorem. The mathematician wants to be able to read and verify the proof. If it is not a theorem, the

mathematician wants to see a counter-example. If the program is unable to determine the truth value of the formula, the mathematician wants to know what *was* accomplished in the proof and what was missing.

The use of a tableau calculus (as opposed to a resolution calculus) makes all of these things possible. In fact, the tableau calculus was invented (in its early form by Gentzen and Beth [57, 23]) in order to simulate human proofs. The calculus invented for IPR is particularly designed to make the output sensible to a mathematician.

1.2.2 Input

In most ATP systems, the user needs to adjust various settings, decide certain indexing schemes on terms and predicates or give the program instructions about how certain information should be used. In order to give this input to the prover, the user often needs to know a lot about the underlying logical calculus being used by the system. Since mathematicians often do not know what first-order logic is, much less the meaning of terms such as "clause," "resolution" or "tableaux," mathematicians have to climb a long and steep learning curve in order to use these systems.

In some systems the user must even enter a proof, usually in a very unnatural language, and the program only checks to see whether it is a correct and complete proof.

The IPR system is designed to avoid these necessities. The user does not need to know about the underlying calculus. The user simply enters the known facts in a theory and the formula to be proved. If IPR does not prove the theorem in a reasonable amount of time, then we get to the topic of the next section.

1.2.3 Interaction

Of course, a theorem prover will be stronger if it enables a human to use expertise to help the prover. This is completely impossible in the case of most really automatic systems. Existing automatic systems translate the problem into some form that is not sensible to a non-expert user. They usually do not attempt to translate an incomplete proof attempt into a form so

that the user can understand what has been done and what remains to be done. Since this is the case, it is practically impossible for the non-expert user to give useful advice or direction to the program during a proof attempt.

The IPR system can be interrupted at any time or put into a mode in which it takes a single step at a time. At each step the program shows the user, in English, what has been done and what remains to be done. It also allows the user to give commands to apply rules in a way that the prover would not have chosen to do on its own.

1.3 A Survey of the Automation of Reasoning

The idea of the automation of reasoning has been around since Euclid. Aristotle, Leibniz, Peano, Boole, Frege, Hilbert, many of the greatest mathematicians and computer scientists of this century including the inventors of the first computers had this dream in mind. For hundreds of years, many great thinkers have envisioned the automation of reasoning and have worked toward the realization of that goal. Most of the historical information in the early part of this section can be found in Martin Davis' survey article [50].

The reasoning algorithms invented by Euclid, such as the division algorithm, are early examples of some types of reasoning that were mechanized by the ancients. Aristotle worked on a system of mechanical rules that he believed would allow their user to draw any true conclusion from a set of assumptions.

Leibniz is famous for inventing the differential and integral calculus. By using this calculus, a machine or person could complete calculations in a few minutes that took Kepler years to figure. Thus, it might be said that human intelligence is no longer required to solve these problems.

Leibniz also developed a "calculus of reasoning". He wanted disputes in human affairs to be answered by what he called "calculators," which were people who knew how the algorithm for reasoning worked. The question would be stated in the formal language and then the disputants would simply say: "let us calculate."

He recognized the three basic elements needed for automatic reasoning: (1) a formal

language, (2) formal rules of inference and (3) knowledge. He worked on a formal, unambiguous language along with a complete set of rules for drawing conclusions (inference rules.)[1] He hoped that the combination of these would allow for any question to be answered finally and unquestionably. His work in this area was left unfinished.

In the nineteenth century, George Boole developed the propositional calculus (in his book *The Laws of Thought* [34]) that provided a language and set of inference rules in which much ordinary common-sense reasoning can be expressed. An advantage to his language was that there was a procedure that would determine whether *any* sentence in the language were true or false in a finite amount of time. As early as 1869, a machine was constructed that could verify statements in propositional logic. The language of propositional logic is, unfortunately, not expressive enough to answer most interesting questions.

In 1879, Gottlob Frege expanded the language to full first-order logic, which allows much more complex statements to be expressed and verified. In fact, the language of first-order logic can be used to express almost any question. Frege introduced some rules of inference but did not develop them fully.

Peano's work was similar to Frege's. He invented a similar formal language and said: "I think that the propositions of any science can be expressed by these signs of logic alone, provided we add signs representing the objects of that science."

Russell and Whitehead showed that this was true. They laid a foundation for all of mathematics (and thus physics and other sciences and fields of reasoning) in logic.

Hilbert was excited by the possibilities and initiated a program of research that had among its goals the discovery of a systematic procedure that would decide the truth or falsity of any statement in Frege's first-order logic. In 1928, Hilbert called the search for this decision procedure the "main problem of mathematical logic" and stated it as follows, "Is it possible to determine whether or not a given statement pertaining to a field of knowledge is a consequence

[1] An *inference rule* is a syntactic rule that is applied to a set of sentences to produce a new sentence that is a logical consequence of the original set of sentences. *Modus ponens* is an example of an inference rule. *Modus ponens* allows us to conclude that the sentence "Q" is a consequence of the two sentences "P" and "P implies Q."

of the axioms?" (See Sections 11 and 12 of Chapter III of *Mathematical Logic* [67].) *Axioms* are simply the most basic assumptions used in a given field of reasoning. So his question could be stated: is there a procedure that can always decide whether a statement is a consequence of some base of knowledge? Given a base of knowledge (which is called a *theory*) a statement that follows logically from that knowledge is called a *theorem* of the theory. Verifying the truth of a statement using inference rules is called proving a theorem.

Hilbert began a program to find a procedure (algorithm) that could decide whether a sentence in first-order logic were true or false given some assumptions. Many logicians (e.g., Gentzen, Beth, Gödel, Church, Turing, Skolem and Herbrand) began working on this problem.

Gentzen and Beth developed algorithms for proving theorems. Their algorithms have the property that they detect if a sentence is true in a finite amount of time. The procedure my system uses is a derivative of their work.

In the 1930s, Church and Turing independently discovered, based on the work of Gödel, that there is no procedure that will decide whether any given statement in first-order logic is true or false. There are, however, procedures that will detect if a statement in first-order logic is true. A procedure with this property is called a *complete* procedure or a *semi-decision* procedure. Unfortunately, complete procedures may never halt if the statement in question is not true; and if the statement is true, there is no telling how long the procedure will take to detect that. So if a complete procedure is followed in an attempt to decide some question, after some time passes, we will not know whether we should give up and guess that the statement is false or whether, if we just continued following the procedure for a few more years, we would find the statement to be true.

But all of these facts apply equally to people and computers. The problem of proving theorems can be difficult and there is no easy answer. Still people do it reasonably well all the time and so there seems to be nothing to prohibit computers from doing it. Many of the great mathematicians and logicians of this century continued to pursue this possibility.

Gödel, in 1944, talking about the difficulties of solving "mathematical problems systematically" said, "But there is no need to give up hope."

The logician and mathematician Hao Wang held the same view.

> The writer, however, feels that the nature and the dimension of the difficulties have been misrepresented through uncontrolled speculation and exaggerated because of a lack of appreciation of the combined capabilities of mathematical logic and calculating machines....
>
> Instead of being discouraged by this, however, one should view it as a forceful reason for experimenting with mechanical mathematics....
>
> It seems, therefore, that the general domain of algorithmic analysis can now begin to be enriched by the inclusion of inferential analysis as a younger companion to the fairly well established but still rapidly developing leg of numerical analysis.
>
> —Hao Wang [117]

Woody Bledsoe had a famous dream. He dreamt of a reasoning machine that would help people solve problems from the mundane to the crucial [27].

Indeed, in the late 1950s and early 1960s, computer implementations of proof procedures began to be reported. In what follows, we will discuss some of the computer theorem provers that have been developed. Some of the techniques mentioned below will be explained in more detail in other parts of this dissertation.

In the last forty years tremendous progress has been made and computers have proved several theorems that have been puzzling mathematicians [89].

Workers in this field approached the problem of automating mathematical reasoning from many different directions. Some wanted to develop methods for first-order logic in general. Since almost any area of mathematics (and almost any other area of reasoning) can be framed in first-order logic, the advantage to this approach is that, if successful, the program could reason in almost any area of mathematics. These programs are called "general-purpose." The framework presented in this dissertation is of this type.

Other workers concentrated on a type of reasoning that is particularly successful in some sub-discipline such as algebra or geometry. Some workers believed that imitating some aspects of the way humans present and consciously discover proofs afforded advantages. Others came up with methods that are difficult for uninitiated mathematicians to understand, yet that

seemed to suit the skill of computers. Some researchers worked on interactive provers that cooperate with the human. These provers accepted hints or proof sketches from the user and used these to prove the given statement.

General-purpose methods. In the 1930s and 1940s, Gentzen [57] and Beth [23] worked on complete systematic proof procedures for first-order logic. These methods are referred to as *sequent* or *tableau* methods. It is a derivative of these methods that is used in the calculus developed in this dissertation and therefore they will be discussed in detail later. Their methods were related to the way humans present proofs. It is not difficult to translate proofs in their systems into readable natural-language proofs (Section 5.2.) Several of the early (late 1950s and early 1960s) automatic theorem proving programs (such as those of Hao Wang [116]) were based on the methods developed by Gentzen and Beth. A method called *model-elimination*, which is related to tableau methods, was developed by Loveland in the early 1960s [83].

Another group of researchers was working on automatic theorem provers based on the work of Herbrand and Skolem. In the early 1960s, J. A. Robinson [104] came up with a new complete procedure for proving theorems in first-order logic called *resolution*. The resolution methods are not so closely related to the way humans present or normally think of proving theorems.

Special-purpose provers. Another vein of work focused on specific types of reasoning or more specialized areas of mathematics such as geometry or number theory. These techniques essentially encode shortcuts that are frequently used in some particular field. The general-purpose provers can prove theorems in these particular fields but it is much more efficient to use the special-purpose shortcuts.

In the late 1950s, H. Gelernter published his work on a theorem prover for statements in geometry [56]. Much later, Shang-Ching Chou wrote a geometry prover that used analytical methods developed by Ritt, Buchberger and Wu [46].

Special provers for elementary number theory and inequality problems were developed based on the work of Presberger and others. Martin Davis implemented such a prover

in 1957 [49]. Larry Hines incorporated systematic procedures for handling certain kinds of inequality problems in 1980 [31, 70].

Various provers were developed for proving theorems by induction. Boyer, Moore and Kaufmann have developed a series of provers for this purpose [38]. Alan Bundy's group developed their own "rippling" techniques for proving theorems by induction [41]. Recently, Baaz, *et. al.* developed the incorporation of induction schemata into the tableau method in a tractable way [7].

Rewriting and equality. Particularly in algebraic problems, a lot of the work in proving a statement involves writing terms in a form that is more useful. This kind of reasoning is called *rewriting*. This is a special area of reasoning in which very successful techniques have been developed.

In 1970, Knuth and Bendix wrote a landmark paper describing a method for proving theorems involving equalities [78]. Supposing that it is known that certain terms are equal, those equalities are automatically used to decide whether some new pair of terms are equal. This technique has been expanded, specialized and generalized by Kapur [76, 124], Bachmair [9] and Dershowitz [51].

Larry Wos and his group incorporated their own rewriting techniques into their resolution provers [122]. These methods are called demodulation and paramodulation.

Another method for handling equality is called the congruence closure technique [85, 92, 105]. This method is complete for ground equality. That is to say that for any set of equalities without variables, and for any theorem that follows from those equalities, this method will prove the theorem using the equalities.

In tableau-based methods, some of these techniques do not work the same way because variables are treated differently in tableaux. Beckert, Gallier, and others have worked on efficient methods for handling equality in tableaux [18, 55].

Set theory and higher-order logic. Reasoning in most mathematical theories (or in most other realms of reasoning) involves reasoning about sets of objects. Reasoning about sets of objects in first-order logic is usually done using a well-known yet complex list of axioms called the axioms of set-theory. Another approach to this kind of reasoning is to use higher-order logics in which it is easier to express statements about sets of objects.

Bledsoe's *Set-Var* method [30] and Bailin and Barker-Plummer's *Z-match* method [10] both used a slight extension to first-order logic in order to allow sets to be expressed and used more easily. Both of these methods discover sets that have the properties needed to prove many statements in set theory and real analysis.

Larry Hines developed a prover for a part of the theory of sets based on the ideas used in his inequality prover (mentioned above) [69]. His techniques handle the transitivity of the subset relation while avoiding the usual complexity that other systematic methods face when using that axiom.

David McAllester's interactive *Ontic* prover checks proofs in set theory using a combination of fast procedures and a language that is based on the theory of English grammar [84].

Quaife used the *Otter* program of Wos and McCune to prove many theorems of set theory based on Gödel's first-order axioms [102].

Peter Andrews developed a completely automatic theorem prover for full higher-order logic [4]. The fact that his prover, TPS, uses full higher-order logic allows many complex statements about sets and functions to be stated more simply than is possible in the language of first-order logic with set-theory.

Human-like techniques. Solow [114] and Polya [98] studied general heuristics used by humans in the theorem-proving process. They studied the basic tactics applied by mathematicians in trying to establish a statement as fact. Alan Bundy advocated attempts to encode some of these human techniques into systematic procedures and computer algorithms [42]. Perhaps the earliest experiment in this direction was conducted by Newell, Simon and Shaw in the late 1950s [93].

Woody Bledsoe, who made substantial contributions to mathematics before dedicating himself to automating reasoning, tried to encode some of his own techniques into his automatic theorem provers [26]. Bledsoe and his collaborators also wanted the programs they developed to be very usable so that the user would not have to be an expert at the underlying proof procedure in order to use the programs [28, 29]. As a result, the provers needed to be able to communicate with people in a natural way.

Interactive provers. Many of the systems mentioned so far were interactive to some extent. Programs that are designed to cooperate with the user are thereby able to prove more theorems than programs that work completely independently of human help. Some programs of this type are called "proof checkers" rather than "theorem provers." In these cases, the user writes out a proof, with or without all of the details, and the prover makes sure that the proof is correct or finds some correct proof using the information entered by the user. These are sometimes called "interactive provers" or "semi-automatic theorem provers." Probably most automated reasoning programs lie somewhere between the extremes of completely automatic theorem proving and proof checking.

Even with all of the work that has been done, the problem of reasoning remains difficult for computers. Hao Wang, a logician who was aware of the theoretical difficulty of the problem of the automation of reasoning, was optimistic. He expected that computers would come into common use as proof assistants for working mathematicians just as they are used today to solve numerical problems [117]. Woody Bledsoe, an expert not only in mathematical analysis but in theorem-proving technology, was optimistic. He dreamt of a reasoning machine that would help people solve problems from the mundane to the crucial [27]. Larry Wos, another expert with a background in mathematics, is optimistic [121]. The programs developed by his group have already solved problems that confounded some of the greatest mathematicians of this century [89].

While great success has been attained, there has been much frustration and disappointment. The lion's share of the realm of reasoning has not been touched by the computer.

One very basic skill that computers have yet to master is that of using successfully an enormous base of knowledge. This is something that humans do rather well. When given a large base of knowledge, most existing programs get bogged down. This problem is addressed by the calculus presented in the body of this dissertation. In short, IPR stores knowledge in a way that is more sensible to humans than clausal form. It selects pieces of knowledge to apply by using strategies. See Section 7.1 for references to some work already existing in this area.

1.4 An Overview of the Dissertation

In Chapter 2, we discuss the components needed for reasoning: a language, inference rules and assumptions. In each case we discuss some of the existing methods and the method that we use in this framework.

In Chapter 3 the rule for applying knowledge to a problem is given, first very informally, then formally. We prove that this rule is correct in that it is impossible to draw a conclusion that is not a consequence of the knowledge using it and that it is capable of proving any true statement. In Section 3.3, we mention the strategies used by the IPR program for selecting theorems to apply during a proof and explain why these strategies are used. These strategies are a key to the success of the framework.

In Chapter 4, we describe the extensions to first-order logic that are implemented in IPR in order to make it easier to get into advanced mathematics. These include some handling of set theory and equality.

Chapter 5 contains a description of the IPR prover that implements the framework described in this dissertation. We describe the interface to the program and we present some technical details about the IPR program and how it works.

In Chapter 6 we give some examples of theorems that IPR has proved including one that, to my knowledge, has not been proved by any other system even though it has been widely distributed.

Chapter 7 contains a survey of closely related work including some that has been done since this calculus was developed. Chapter 8 contains the conclusion of the work and ideas for

future research that will improve the reasoner and the interface.

Chapter 2

Formal Reasoning

> "Contrariwise," continued Tweedledee, "if it was so, it might be; and if it were so, it would be; but as it isn't, it ain't. That's logic."
>
> —Lewis Carroll [44]

As was mentioned in Section 1.3, and indeed was known by Leibniz, in order to automate reasoning we need a language, inference rules, and some sentences in the language that we assume are true. These assumptions are called axioms and definitions and will be stored in the knowledge base. In this chapter, we discuss issues relating to the selection of language, inference rules and representation of assumptions.

Consider an example of reasoning about real numbers. We have a **language** in which expressions such as $2, \pi, 4 + e$ and x^2 are terms. Some sentences (which may or may not be true) in the language of the reals are:

$$2 < 4$$
$$16^e = \log_3 4$$

We also use some special **rules of inference** when dealing with the real numbers. If we know that $x < y$ and that $y < z$, then we can conclude that $x < z$. This inference rule can also be considered an axiom. The same is true of most inference rules that are special to a particular field of reasoning. The special inference rules are really just shortcuts for using axioms along with the general inference rules that we will present in Section 2.2.

Finally, in the theory of the reals, we have certain **assumptions** or **knowledge** such as the fact that between any two distinct real numbers there is a third distinct real number.

When a program turns a theorem or axiom into an inference rule, that program is called a *special-purpose* prover. In this project, we want to stay general-purpose. (See Section 1.3.) Here we would like to use a language and a set of inference rules such that we can reason about almost anything rather than only about the real numbers. In order to reason in special areas, such as the theory of the reals or topology, we will use the appropriate assumptions about objects in that theory. We will draw conclusions from those assumptions using general purpose inference rules.

In Section 2.1 we discuss the choice of general purpose language and define the language of first-order logic. Once we have a language, we will need a method for establishing the truth of sentences in the language. In Section 2.2 we mention three different sets of inference rules, each of which is complete in the sense defined in the introduction. We will discuss advantages of each and why we chose the one we did.

Finally, in Section 2.3, we describe how assumptions in the language are stored in the knowledge base. We will also mention some advantages to using this format. We argue that the format for storing knowledge is useful in the reasoning process partly due to the fact that it resembles the way humans write and prove theorems more closely than more common computer oriented formats. In Chapter 3, we will describe how the knowledge is selected and used to prove new statements.

2.1 Language for a Theory

The field of mathematics is a good source of problems in knowledge representation and reasoning. Many have searched for a language that is expressive and suitable for use in efficient automated reasoning in mathematics. Most of the present work was done in the context of a language called first-order logic. Before defining first-order logic, we briefly mention some other languages that have been proposed and used by others.

2.1.1 Language Issues

The two issues faced when selecting a language are expressiveness and complexity of reasoning. Languages that are very expressive are accompanied by very complex inference systems, while languages that allow efficient inference are not very expressive. We use first-order logic because it balances these two issues relatively well.

The simplest solution to the problem of the complexity of the reasoning is to restrict the expressiveness of the language in various ways. This solution does not seem satisfactory for the purposes of proving theorems in classical mathematics because less expressive languages are not able to make many interesting statements in mathematics. For example, if the language is restricted to Horn clauses, then reasoning can be very efficient. In this case, there are procedures that are guaranteed not to allow certain kinds of complexity to arise in a proof. However, in the language of Horn formulas it is very difficult to express many interesting facts in mathematics including many very basic definitions. Propositional logic is another example of a language in which reasoning is very efficient. In propositional logic, there is a decision procedure. But, in propositional logic, it is impossible even to express the notion of "every" or "some" in a general way.

David McAllester [58, 59, 86, 87] has developed languages that are more expressive than first-order logic and for which the subset of the language for which there is a decision procedure properly contains the subset of first-order logic for which there is a decision procedure. This is a great advantage. The disadvantages include the unfortunate fact that the languages are not well-known and the multiplicity of inference rules that must be used in the languages. McAllester's languages are unfamiliar not because they are any more difficult to comprehend or read; in fact, they are easier to read than first-order logic since they are more closely related to English grammar.

Jaakko Hintikka advocates the use of "independence-friendly" logic since it is more expressive than ordinary first-order logic [72]. Naturally, reasoning in this logic is more complex than in ordinary first-order logic. Additionally, Hintikka's independence-friendly logics do not obey the law of the excluded middle.

Higher-order languages are much more expressive than first-order logic. In these languages, it is easier to express statements about properties, functions and sets. Reasoning about sets of objects using first-order logic is not simple. Frequently, sentences that can be expressed in first-order logic with the axioms of set theory can be expressed in higher-order logic with less quantification. A disadvantage to higher-order logic is that reasoning is much more complex. In higher-order logics, the branching factor—that is, the number of choices one has in taking the next step in the proof—can be infinite.

2.1.2 First-Order Logic

First-order logic is defined in many textbooks [90, 113]. In this dissertation we will remain rather informal in the definition of first-order logic, particularly regarding semantics. To see all of the details, one should read the appropriate articles and books mentioned in the bibliography [113]. In Section 5.2.1 we give a slightly more formal definition of the syntax for the language that is input to the IPR program.

First we will describe *terms* in the language. Terms are expressions that stand for objects in the universe of discourse. Terms in first-order logic correspond to nouns in English. For example, in the theory of real numbers, $2, \pi, e$, and $\log_2(3)$ are terms; they stand for real numbers. Variables are also terms and they are thought of as ranging over the universe of discourse. For example, the expression $x^2 + 1$ is a term in the theory of real numbers and it could stand for any real number greater than or equal to 1. ($x^2 + 1 \geq 1$ because otherwise $x^2 < 0$, which is impossible.) Terms can be constructed from smaller terms by the use of *function symbols*. For example, cos is a function symbol in the theory of the reals that takes one argument. For any term t, we can construct a term $\cos(t)$ where t is the argument of the function cos. All arguments given to function symbols must be terms. In general, if f is a function symbol that takes n arguments (where n is a positive integer) then $f(t_1, \ldots, t_n)$ denotes a term if $t_1, \ldots, t_n$ are terms. Another example is $+$: given any two terms, s and t, we can construct the expression, $+(s, t)$, (usually denoted $s + t$) which is another term in the language. Here s and t are called *arguments* of the function symbol $+$. If a function symbol, f, takes no arguments then we call it a *constant term*. Examples of function symbols in the theory

of topology[1] are top-to-class, which takes one argument, and the-set-inverse-image, which takes two arguments. Hence, for example, top-to-class(S) and the-set-inverse-image(f, G) are terms in the language of topology.

The other basic element of the language is the *predicate*. Predicates construct sentences that may be true or false. Just like function symbols, *predicate symbols* can take arguments. All arguments given to predicate symbols must be terms in the language. For example, $<$ is a predicate symbol in the theory of the reals and $<(2, \pi)$ (usually denoted $2 < \pi$) is a (true) predicate in the language. In logic, the word *formula* is used to refer to an expression that may be true or false. A formula that consists entirely of a single predicate symbol and its arguments is called an *atomic formula*. Therefore, $2 < \pi$ is an atomic formula in the language of the reals.

We can build more complex formulas out of basic ones using the following *logical connectives*.

$\neg$ **"not"** If A is a formula, then $\neg A$ is a formula that is true when A is false.

$\wedge$ **"and"** If A and B are formulas, then $A \wedge B$ is a formula that is true when A and B are both true.

$\vee$ **"or"** If A and B are formulas, then $A \vee B$ is a formula that is true when one of A or B is true.

$\supset$ **"implies"** If A and B are formulas, then $A \supset B$ is a formula that is true when A is false or when B is true.

$\leftrightarrow$ **"if and only if"** If A and B are formulas, then $A \leftrightarrow B$ is a formula that is true when A and B are both true or both false.

[1] Throughout this dissertation, we will be illustrating the methods we introduce by showing how they are used to establish the truth of statements in particular mathematical theories. Most of these examples will come from the mathematical theory called topology. Topology is the study of sets with a certain structure and functions between such sets. If you are not familiar with the particular field of reasoning, such as topology, just try to follow how the *reasoning* is being done and don't worry about what the sentences *mean*. In other words, you will only need to be able to see that a certain sentence is a logical consequence of other sentences and you will not need to understand what any of the sentences mean.

$\forall$ **"for all"** If A is a formula and x is a variable, then $(\forall x)A$ is a formula that is true when A is true regardless of what element of the universe of discourse x is replaced with in A.

$\exists$ **"for some"** If A is a formula and x is a variable, then $(\exists x)A$ is a formula that is true when there is some element of the universe, a, such that if x is replaced by a in A then the resulting formula is true.

Any occurrence of the variable x in a formula of the form $(\forall x)A$ or $(\exists x)A$, is called a *bound* occurrence of x. The variable x is said to be *bound* in those formulas. Any occurrence of a variable that is not bound is called *free*. A formula with no free variables is called a *closed formula* or a *sentence*.

There are technical points about renaming bound variables when free variables are replaced by other terms. The concerned reader should refer to a textbook on first-order logic for more rigorous definitions [113, 54].

A *literal* is an atomic formula or the negation of an atomic formula. Thus, a literal can contain no logical connectives other than one occurrence of $\neg$. If L is a literal, then we define the *complement*, $\overline{L}$, of L as follows:

$$\overline{L} = \begin{cases} \phi & \text{if } L = \neg\phi \\ \neg L & \text{otherwise.} \end{cases}$$

For example, if open-in is a predicate symbol that takes two arguments and G and T are variables, then $\overline{\text{open-in}(G,T)} = \neg\text{open-in}(G,T)$ and $\overline{\neg\text{open-in}(G,T)} = \text{open-in}(G,T)$. We will say that L and $\overline{L}$ are *complementary* literals.

We will also use the atomic formula, $\bot$, which will always be false and the atomic formula, $\top$, which will always be true.

Example 2.1 Here we give examples of the way formulas are formed. We illustrate this with one of the axioms of the theory of topology. We will start with atomic formulas and build up more complex formulas using the rules just introduced. Topology is a field of mathematics that has its own special assumptions. Here are some of the predicate symbols in the language of

topology: open-in, which takes two arguments, and function-from-to and continuous-from-to, which both take three arguments. Each of the following expressions are atomic formulas.

$$\text{open-in}(G, T) \tag{2.1}$$

$$\text{open-in}(\text{the-set-inverse-image}(f, G), S) \tag{2.2}$$

$$\text{function-from-to}(f, \text{top-to-class}(S), \text{top-to-class}(T)) \tag{2.3}$$

$$\text{continuous-from-to}(f, S, T) \tag{2.4}$$

Notice that none of these atomic formulas are closed formulas. That is, each of them has some free variables occurring in it.[2]

Now we will construct non-atomic formulas from formulas 2.1 and 2.2.

$$\text{open-in}(G, T) \supset \text{open-in}(\text{the-set-inverse-image}(f, G), S) \tag{2.5}$$

Formula 2.5 is not an atomic formula because it is formed from two atomic formulas and the implication symbol. Note that each of the variables, f, G, T and S occur free in formula 2.5.[3]

$$(\forall G)(\text{open-in}(G, T) \supset \text{open-in}(\text{the-set-inverse-image}(f, G), S)) \tag{2.6}$$

In formula 2.6, G is now a bound variable and f, T and S are the only free variables.[4] From formulas 2.6 and 2.3 we construct a more complex formula.

$$\begin{array}{c}\text{function-from-to}(f, \text{top-to-class}(S), \text{top-to-class}(T)) \wedge \\ (\forall G)(\text{open-in}(G, T) \supset \text{open-in}(\text{the-set-inverse-image}(f, G), S))\end{array} \tag{2.7}$$

And now we take one more step, adding formula 2.4.[5]

$$\begin{array}{c}\text{continuous-from-to}(f, S, T) \leftrightarrow \\ (\text{function-from-to}(f, \text{top-to-class}(S), \text{top-to-class}(T)) \wedge \\ (\forall G)(\text{open-in}(G, T) \supset \text{open-in}(\text{the-set-inverse-image}(f, G), S)))\end{array} \tag{2.8}$$

[2] Formula 2.1 means that G is open in the topology T. (Again, if you don't know what this means, just pay attention to the way formulas are formed.) Formula 2.2 means that $f^{-1}(G)$ is open in the topology S. Formula 2.3 means that f is a function from the class associated with the topology S to the class associated with the topology T. Formula 2.4 means that f is a continuous function from S to T where S and T are topological spaces.

[3] Formula 2.5 means that if G is open in T then $f^{-1}(G)$ is open in the topology S.

[4] Formula 2.6 is intended to be true only if for any open subset G of T, $f^{-1}(G)$ is open in the topology S.

[5] Formula 2.7 is true only if f is a function from the class associated with the topology S to the class associated with the topology T and for any open subset G of T, $f^{-1}(G)$ is open in the topology S.

Formula 2.8 still has three free variables. We remedy that situation now.

$$\begin{array}{c}(\forall f)(\forall S)(\forall T)(\text{continuous-from-to}(f,S,T) \leftrightarrow \\ (\text{function-from-to}(f,\text{top-to-class}(S),\text{top-to-class}(T)) \wedge \\ (\forall G)(\text{open-in}(G,T) \supset \text{open-in}(\text{the-set-inverse-image}(f,G),S))))\end{array} \tag{2.9}$$

Now formula 2.9 is a closed formula or a sentence. The closed formula 2.9 is an axiom of the theory of topology.[6]

Formula 2.9 is a special kind of axiom called a predicate definition. A predicate definition is a sentence that introduces a new predicate to the language, continuous-from-to in this case, as an abbreviation for a formula already in the language. Definitions are axioms that we are allowed to assume to be true without danger of introducing a contradiction into a theory [22].

We will learn, in Section 2.3, how such a formula can be stored in a computer knowledge base in a form that is useful and, in Section 3.2, how this knowledge can be used in the process of trying to prove other statements in the theory of topology. In Section 3.3, we discuss systematic strategies for selecting knowledge from the knowledge base.

The implementation, IPR, of the framework presented in this dissertation uses a language slightly more expressive than first-order logic. The extensions are described in Section 4.2.

2.2 Reasoning in a Theory

Now that we have selected an expressive language, we need rules of inference for determining which sentences in the language are true and which sentences are logical consequences of other sentences.

A proof is a structure that can be built in a finite amount of time using syntactic rules. In the present case, we follow a procedure by applying certain syntactic rules to the sentence we want to prove and by this we build a structure. If the structure we build has a

[6] The formula is intended to mean that for any function f and topological spaces S and T, f is a continuous function from S to T if and only if f is a function from the class associated with the topology S to the class associated with the topology T and for any open subset G of T, $f^{-1}(G)$ is open in the topology S.

certain property (if it is *closed*) then we say that the structure is a proof of the sentence. If the procedure we use is *sound* then the existence of a proof implies that the sentence is true. If the procedure is *complete*, then for any true sentence, we can find a proof by this procedure.

There are many sets of inference rules that are adequate for this task. The two methods we present, the sequent calculus (Section 2.2.2) and the tableau calculus (Section 2.2.1), are very closely related. In fact, the latter can be thought of as a more efficient version of the former [113]. The reason we present both is merely that some issues are easier to explain in terms of the sequent calculus even though it is less efficient to use. The technical details and most examples will be presented in terms of tableaux while the rules for breaking knowledge into the form in which it is stored are given in sequent form.

The sequent calculus was developed by Gentzen [57] for the purpose of formalizing the proofs that humans write. The system was intended to be very similar to natural deduction. As a result, it is relatively easy to translate proofs from this system into natural language. Further, it is not too difficult for a human to understand an incomplete proof in this system.

Resolution is a third widely used calculus for reasoning in first-order logic. In order to apply resolution, the negation of the sentence to be proved must first be translated into a conjunction of formulas in clausal or *Kowalski* form. A formula in Kowalski form is basically a sequent in which all of the formulas in the hypotheses and in the goals are atomic formulas [42, 45]. A formula in Kowalski form is usually called a *clause* and when a formula is translated into a conjunction of clauses, it is said to be in *clausal* form. Once the negation of a formula is translated into this form, the resolution inference rule is applied in order to find a proof of the original formula [45]. We do not give the details of the resolution procedure here. The interested reader should see the apropriate references in the bibliography [45, 122]. In Section 2.3.1, we present the procedure for translating a formula into clausal (or Kowalski) form.

2.2.1 Tableaux

Most existing tableau calculi can be divided into two classes: analytic and clausal. IPR's method is a hybrid so an understanding of each will be useful in understanding IPR's method.

IPR uses a non-clausal form for storing knowledge. The novel tableau calculus presented in Section 3.2 can just as well be applied in the context of clausal tableaux.

The proof procedure of the analytic tableau was developed by Smullyan [113] based on the work of Beth [22] and Hintikka [71]. The method was extended to allow for free variables by Prawitz [100]. See Fitting's book [54] for a formal and rather up-to-date treatment of the procedure. We do not present all of the details here. We present the signed version in a novel way for pedagogical reasons, then we describe how to use the more common unsigned version.

The model elimination calculus [83] was the first manifestation of a clausal tableau calculus. More recently, the theory of clausal tableaux has been studied more thoroughly [81]. The essential difference is that an analytic tableau is built by breaking an initial formula into pieces. The essential feature of clausal tableaux is that each formula on a branch is a literal occurring in the input clause set that is formed from the input formula as a preprocessing step.

Analytic Tableaux

We introduce analytic tableaux with signed formulas first for pedagogical reasons. In the rest of the paper, we will use the unsigned version.

To prove a formula ϕ using the signed analytic tableau method, we will build a signed tableau (table or tree) of signed formulas rooted at the signed formula $\mathbf{Sh}\ \phi$. This signed formula can be pronounced "show ϕ." The other sign is $\mathbf{Sp}$ and $\mathbf{Sp}\ \phi$ would be pronounced "suppose ϕ."[7] We now give a meaning to the complement of a signed formula: if ϕ is a formula, then $\overline{\mathbf{Sh}\ \phi} = \mathbf{Sp}\ \phi$ and $\overline{\mathbf{Sp}\ \phi} = \mathbf{Sh}\ \phi$. For any signed formula Φ, we say that Φ and $\overline{\Phi}$ are complementary.

A branch of a tableau is *closed* if it contains a complementary pair of formulas or if it

[7] In the ordinary presentation of the signed tableaux, $\mathbf{T}$ is used rather than $\mathbf{Sp}$ and $\mathbf{F}$ is used rather than $\mathbf{Sh}$.

contains one of the formulas $\neg\top$ or $\bot$. A tableau is closed if every branch is closed. When we find a closed analytic tableau rooted at **Sh** ϕ, we say that it is a proof of ϕ.

We build the signed tableau down from the initial signed formula using the inference rules below until all branches of the signed tableau are simultaneously closed.

There are five basic inference rules, called α, β, δ and γ, one for each of the four classes of non-atomic signed formulas listed here as well as the tableau substitution rule described later.

α	β	δ	γ
Sh $(A \supset B)$	**Sp** $(A \supset B)$	**Sp** $(\exists x)A$	**Sh** $(\exists x)A$
Sp $(A \wedge B)$	**Sh** $(A \wedge B)$	**Sh** $(\forall x)A$	**Sp** $(\forall x)A$
Sh $(A \vee B)$	**Sp** $(A \vee B)$		
Sh $\neg A$			
Sp $\neg A$			

When an α-formula, for example, occurs on a branch of the signed tableau then the α-rule (given below) can applied to the formula. When any of the inference rules is applied to a formula, some new formulas are added to the signed tableau. We will use the following abbreviations for the formulas added to the signed tableau.[8]

α	α_1	α_2
Sh $(A \supset B)$	**Sp** A	**Sh** B
Sp $(A \wedge B)$	**Sp** A	**Sp** B
Sh $(A \vee B)$	**Sh** A	**Sh** B
Sh $\neg A$	**Sp** A	
Sp $\neg A$	**Sh** A	

β	β_1	β_2
Sp $(A \supset B)$	**Sp** B	**Sh** A
Sh $(A \wedge B)$	**Sh** A	**Sh** B
Sp $(A \vee B)$	**Sp** A	**Sp** B

δ	$\delta(t)$
Sp $(\exists x)A$	**Sp** A^x_t
Sh $(\forall x)A$	**Sh** A^x_t

γ	$\gamma(y)$
Sh $(\exists x)A$	**Sh** A^x_y
Sp $(\forall x)A$	**Sp** A^x_y

Now we will present the inference rules using this notation. In each of the inference rules there is a horizontal line over which is a formula that is thought of as being a formula on a branch of the current signed tableau. The formula or formulas below the line are added to the branch containing the original formula. If more than one formula is listed horizontally below the line, then more than one branch is added to the signed tableau. The formulas in a vertical list below the line are all added to the existing branch.

[8] The notation A^a_b represents the expression obtained by replacing every free occurrence of the variable a in the formula A with the term b.

Inference Rule 2.1 (α-rule)

$$\frac{\alpha}{\begin{array}{c}\alpha_1\\ \alpha_2\end{array}}$$

Inference Rule 2.2 (β-rule)

$$\frac{\beta}{\beta_1 \mid \beta_2}$$

Inference Rule 2.3 (δ-rule)

$$\frac{\delta}{\delta(f(\vec{y}))}$$

where $\vec{y}$ is the vector of free variables in δ and f is a new Skolem function symbol.

Inference Rule 2.4 (γ-rule)

$$\frac{\gamma}{\gamma(y)}$$

where y is a new free variable.

With the exception of the γ-rule, each of these rules is typically applied only once to each appropriate formula. For example, the β-rule is typically only applied once to any given β-formula. After the first application, the β-formula is considered *used*. On the other hand, the γ-rule is typically applied any number of times to the same formula.

Example 2.2 We show how to use these rules to prove the propositional formula

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

As we mentioned above, to prove a formula ϕ using the signed tableau method, we will build a signed tableau of formulas rooted at the formula **Sh** ϕ. We build the signed tableau up from that formula using the inference rules above. Figure 2.1(A) shows the signed tableau built from this formula. No more rules may be applied to an unused formula on this signed tableau. Each edge in the signed tableau is labeled with the rule that was applied to create the node below it.

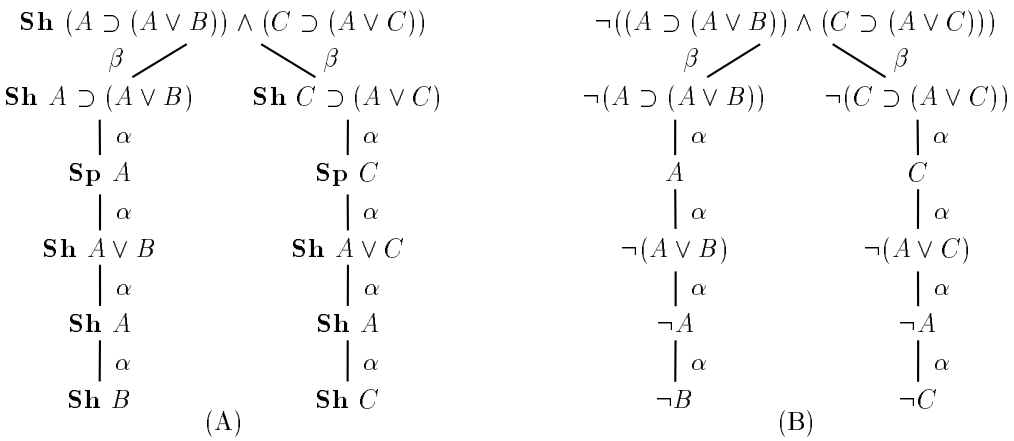

Figure 2.1: The signed tableau (A) and ordinary tableau (B) for Example 2.2.

Let us examine in detail how this tableau was constructed. Since we want to prove the conjunction, we need to prove each conjunct: one on each branch. On the left branch, we want to show that A implies $A \vee B$ so we can suppose A and try to show $A \vee B$. That is, we can show A or B. The right branch is similar.

Now notice that each branch contains a complementary pair of formulas. Therefore the tableau is closed and we have a proof of the formula we started with. The fact that the tableau procedure is sound [54] tells us that that formula must be true.

The proofs found by the tableau procedure are intended to be comprehensible. How do we follow this proof? Look at one branch at a time. On the first branch we are supposing that A is true on that branch and A becomes one of our goals. We can obviously conclude A from the assumption that A is true. Therefore, we say that that branch is closed, i.e., that branch of the proof is finished. The same can be done on the right branch with **Sp** C and **Sh** C.

Example 2.2 demonstrates the basics of how theorems are proved using the signed tableau method. But things get more complicated when variables are involved.

Example 2.3 Now we prove the formula

$$(\forall x)(\forall y)(P(x) \vee Q(y)) \supset ((\forall x)P(x) \vee (\forall y)Q(y)).$$

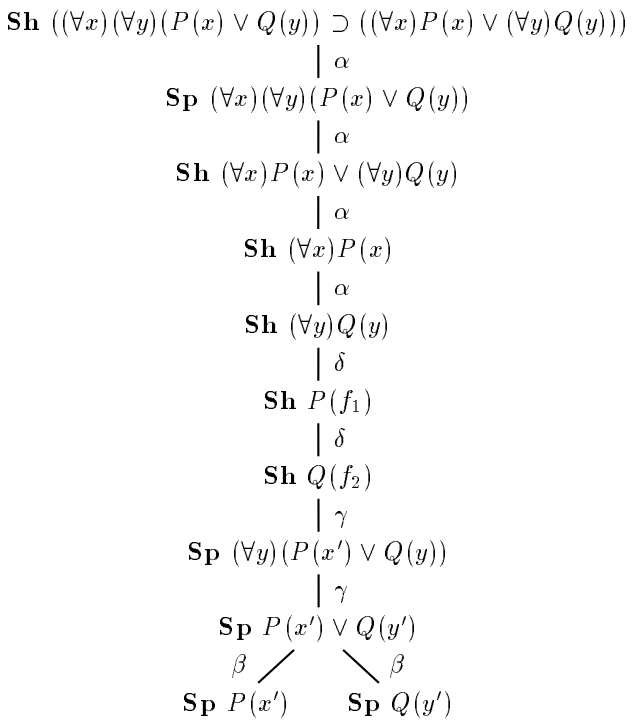

Figure 2.2: The signed tableau for Example 2.3. The symbols f_1 and f_2 are Skolem constants.

We can construct the signed tableau in Figure 2.2. We are allowed to continue to expand the signed tableau by applying the γ-rule any number of times but this will never allow us to close either branch of the signed tableau. So we introcude a new rule (the tableau substitution rule below) that allows us to close a branch by applying a substitution of variables for terms. In this case, since x' and y' are free variables, we can close the signed tableau by replacing x' with f_1 and y' with f_2. After this, the bottom of the signed tableau becomes the signed tableau in Figure 2.3, which *is* closed.

Example 2.3 motivates the definition of a new inference rule that allows us to substitute terms for free variables when doing so closes the tableau or a branch. We must refer the reader elsewhere for the technical details of this definition [54].

Inference Rule 2.5 (Tableau Substitution) If T is a properly constructed tableau and $T\sigma$ is the result of applying the substitution σ to every formula in T, then $T\sigma$ is a properly constructed tableau.

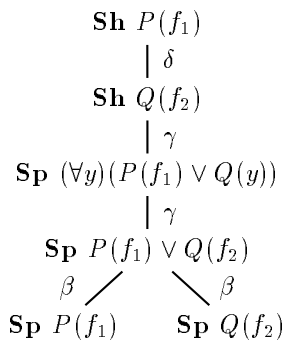

Figure 2.3: The bottom of the instantiated signed tableau for Example 2.3. The symbols f_1 and f_2 are Skolem constants.

So the complete analytic tableau calculus is the combination of the α-, β-, γ-, δ- and tableau substitution rules. For completeness, we must have a procedure for applying these rules that obeys certain fairness restrictions. In particular, the α-, β- and δ-rules must be applied at least once to each appropriate formula, the γ-rule must be applied infinitely many times to any γ-formula and the tableau substitution rule can be applied using a most general substitution that closes the entire tableau. It is this procedure that is proved complete and sound elsewhere [113, 54].

This completes the description of the analytic tableau method. The method is complete, i.e., any true sentence can be proved by the method [54], and there are various methods of making it more efficient [8, 20, 82, 94].

We emphasize how easy it is to translate proofs in this system into natural, readable proofs. Section 5.2 gives some examples of proofs translated automatically into English by the IPR program.

Now that the basic method is understood, we want to introduce the more common way of using tableaux, that is the unsigned method. It is the same as what has gone before except that every **Sh** should be replaced by a $\neg$ and every **Sp** should be omitted. The reader will notice that one instance of the α-rule becomes useless because it only translates the formula $\neg A$ (**Sp** $\neg A$) into $\neg A$ (**Sh** A.)

The unsigned version of the tableau for Example 2.2 is given in Figure 2.1(B). Hence-

forth, we will deal with unsigned tableaux. Furthermore, we will not label edges with inference rules except in the case of the new inference rules introduced later or in cases when there might be confusion.

Clausal Tableaux

We follow Letz' treatment of clausal tableaux [81]. We refer the reader to Letz' work for more details.

A *literal* is an atom or the negation of an atom. A *clause* is a disjunction of literals. A *clause set* is a conjunction of clauses. To prove a formula by the clausal tableau method, the formula is negated and translated into clausal form by Skolemizing [45]. (Translating to clausal form is treated in detail in Section 2.3.)

A clausal tableau is initialized by placing the formula $\top$ at the initial node. From the input clause set a clausal tableau is constructed by adding clauses (with free variables possibly renamed not to conflict with free variables already on the tableau) to the tableau. Each literal in the input clause is used to extend a new branch. Therefore, a unit clause (a clause consisting of a single literal) does not cause a splitting in the tableau but any other clause (excluding the empty clause) does cause a splitting.

A clausal tableau is closed in the same way that an analytic tableau is closed.

A clausal tableau is *connected* if each inner node N labeled with a literal L has a leaf node N' among its immediate successor nodes that is labeled with the literal $\overline{L}$. A tableau is *regular* if no two nodes on the same branch are labeled with the same formula.

2.2.2 Sequent Calculus

The sequent calculus was described in 1935 by Gerhard Gentzen [57]. In 1968, Raymond Smullyan published a beautiful unification of Gentzen's work with the analytic tableau method [113]. We will use some of Smullyan's notation here. We are also using further extensions to the system that incorporate free variables [54, 100].

A *sequent* consists of two sets of formulas, denoted $U \rightarrow V$, where U and V are sets

of formulas. We call U the *hypotheses*, *antecedents* or *suppositions* and we call V the *goals*, *succedents*, *consequents* or *conclusions*. If one of U or V are empty, then we simply write $\rightarrow V$ or $U \rightarrow$, respectively. If A and B are formulas then we use the following abbreviations.

$$\begin{aligned} A \rightarrow B \quad &\text{for} \quad \{A\} \rightarrow \{B\} \\ A, U \rightarrow V \quad &\text{for} \quad \{A\} \cup U \rightarrow V \\ U \rightarrow B, V \quad &\text{for} \quad U \rightarrow \{B\} \cup V \end{aligned}$$

Suppose $U = \{A_1, \ldots, A_n\}$ and $V = \{B_1, \ldots, B_m\}$. The sequent $U \rightarrow V$ is *true* or *valid* if the formula $(A_1 \wedge \cdots \wedge A_n) \supset (B_1 \vee \cdots \vee B_m)$ is true. In other words, a sequent is thought of as saying that the conjunction of the hypotheses implies the disjunction of the goals.

To prove a formula ϕ, we will build a tree of sequents rooted at the sequent $\rightarrow \phi$. We build the tree up from that sequent using the inference rules below until all leaves of the tree are closed by the following definition.

Axiom Schema 2.6 Any sequent of one of the forms

$$A, U \rightarrow A, V \qquad \bot, U \rightarrow V \qquad U \rightarrow \top, V$$

is closed.

If a sequent is not closed by this axiom, then we need inference rules to apply to it in order to reduce it to sequents that are closed by the axiom (if possible.)

In each of the inference rules below there is a horizontal line under which is a sequent that is thought of as being a leaf of the current sequent tree. The sequent or sequents above the line are added to the branch containing the original sequent so that the original sequent is no longer a leaf. If more than one sequent is above the line, then more than one branch is added to the tree. Γ and Δ represent sets of formulas.

Inference Rules 2.7 (α-rules)

$$\frac{\Gamma, A \rightarrow \Delta, B}{\Gamma \rightarrow \Delta, A \supset B} \qquad \frac{\Gamma, A, B \rightarrow \Delta}{\Gamma, A \wedge B \rightarrow \Delta} \qquad \frac{\Gamma \rightarrow \Delta, A, B}{\Gamma \rightarrow \Delta, A \vee B}$$

$$\frac{\Gamma, A \rightarrow \Delta}{\Gamma \rightarrow \Delta, \neg A} \qquad \frac{\Gamma \rightarrow \Delta, A}{\Gamma, \neg A \rightarrow \Delta}$$

Inference Rules 2.8 (β-rules)

$$\frac{\Gamma, B \to \Delta \mid \Gamma \to \Delta, A}{\Gamma, A \supset B \to \Delta} \qquad \frac{\Gamma \to \Delta, A \mid \Gamma \to \Delta, B}{\Gamma \to \Delta, A \wedge B} \qquad \frac{\Gamma, A \to \Delta \mid \Gamma, B \to \Delta}{\Gamma, A \vee B \to \Delta}$$

Inference Rules 2.9 (δ-rules)

$$\frac{\Gamma \to \Delta, A^x_{f(\vec{y})}}{\Gamma \to \Delta, (\forall x) A} \qquad \frac{\Gamma, A^x_{f(\vec{y})} \to \Delta}{\Gamma, (\exists x) A \to \Delta}$$

where $\vec{y}$ is the vector of free variables in $(\forall x)A$ and f is a new Skolem function symbol.

Inference Rules 2.10 (γ-rules)

$$\frac{\Gamma, A^x_{x'} \to \Delta}{\Gamma, (\forall x) A \to \Delta} \qquad \frac{\Gamma \to \Delta, A^x_{x'}}{\Gamma \to \Delta, (\exists x) A}$$

where x' is a new free variable.

So we have a rule to apply to any non-atomic formula in any position in a sequent. The γ-rules are those that introduce new free variables. The δ-rules introduce new Skolem functions. Among the propositional rules are the α-rules that do not cause branching in the tree and the β-rules that do cause branching. In the α-rules, the formula to which the rule is applied (e.g., $A \supset B$ in the goals) is called an α-formula and similarly for β, δ and γ.

Inference Rule 2.11 (Tree Substitution) If $\mathcal{T}$ is a properly constructed sequent tree and $\mathcal{T}\sigma$ is the result of applying the substitution σ to every formula in $\mathcal{T}$, then $\mathcal{T}\sigma$ is a properly constructed sequent tree.

A branch of a sequent tree is called *closed* if its leaf is closed by Axiom Schema 2.6. A sequent tree is closed if all of its branches are closed. A formula ϕ is proved if there is a tree of sequents built by these rules based at $\to \phi$ such that each leaf is closed. If a formula, ϕ, is proved in this way, then the formula is true [113, 54].

The relationship between the semantic tableau method and the sequent calculus is close. The formulas in the hypotheses of the sequent correspond to positive formulas on a branch of a tableau (or the **Sp** formulas on a signed tableau) and formulas in the goals of a sequent correspond to negative formulas on a branch of a tableau (or the **Sh** formulas on a

signed tableau.) The unused formulas on any path of a tableau starting at the root correspond to a sequent whose hypotheses are the positive formulas on the path and whose conclusions correspond to the negative formulas on the path. Notice also that Axiom Schema 2.6 corresponds to the definition of a closed tableau. Just as is the case for tableaux, we can apply a substitution across an entire sequent tree in order to apply Axiom Schema 2.6.

We will primarily use the sequent calculus to describe the formation of sequents in the knowledge base except that some of the rules will be changed for that purpose. This is detailed in Section 2.3.

2.3 Knowledge in a Theory

> Central in all of this is *knowledge*. We see no satisfactory solution to the ATP [Automated Theorem Proving] performance problem without a substantial *knowledge base*, which contains much known mathematics...
>
> —Woody Bledsoe [33]

The IPR system is designed with a specific application or challenge in mind. That is the $n + 1$ challenge: enter the first n theorems, definitions and axioms from a textbook and let the system prove the next theorem. Obviously, a system that could succeed at this problem for many n, should be useful in proving open problems. Also, this design seems to be something that a mathematician or student of mathematics might find useful. (This challenge is discussed in more detail in Section 8.3.)

Some systems are built with the intention of going back to a minimal, independent set of axioms. This approach makes some problems easy and is quite appropriate for certain applications (such as logic programming) but is not appropriate for the $n + 1$ problem. For one thing, the user who wants to prove the next theorem does not know exactly which axioms will or will not be needed in a proof. There may be certain axioms of equality, set theory, and various advanced mathematical axioms that may or may not be needed for the proof of the next theorem. Another problem is that even if a minimal set of axioms were known to be

sufficient to prove a certain theorem, a proof built from those axioms will be much longer and much less readable than a proof that uses the knowledge that has been built up in the theories.

In advanced theories in real applications, there will be unneeded knowledge and information other than a set of independent axioms. The presence of unneeded information tends to make many systems (particularly resolution systems) lose a performance edge.

In most automated theorem proving systems, the axioms, theorems and definitions of a theory are stored in clausal form. The advantages to clausal form have to do with the fact that procedures using this form can take various shortcuts without losing completeness. We describe the translation to clausal form and then discuss some disadvantages to this. We then present a non-clausal form that has some advantages (and, of course, disadvantages.) The IPR prover uses the non-clausal form presented in Section 2.3.2.

Example 2.4 Before we get to the details, consider the definition of a continuous function that we discussed earlier in Example 2.1. This is a piece of knowledge that we want the prover to use when it attempts to find a proof of other statements in the theory of topology.

A function $f : S \to T$ *is* continuous *if and only if* f *is a function from* S *to* T *and for every open set* $G \subset T, f^{-1}(G)$ *is open in* S.

We will store it in the prover's knowledge base when it searches for proofs of theorems in topology. This theorem contains the following three pieces of information:

- *if* $f : S \to T$ *is a continuous function then* f *is a function from* S *to* T,
- *if* $f : S \to T$ *is a continuous function and* G *is an open subset of* T, *then* $f^{-1}(G)$ *is open in* S *and*
- *if* f *is a function from* S *to* T *and for every open set* $G \subset T, f^{-1}(G)$ *is open in* S, *then* $f : S \to T$ *is continuous.*

The rules we present in Section 2.3.2 break the definition into exactly these three bits of information (Example 2.6.) The fact that theorems are stored in a form that makes sense logically and cognitively gives the framework some apparent advantages.

- It allows for an intuitive interface since the user can understand the contents of the knowledge base. See Section 5.2.3.
- It allows the algorithm that selects theorems from the knowledge base to implement methods that seem to resemble some of the heuristics used by human mathematicians. See Section 3.3.

Axioms, theorems and definitions are stored in the knowledge base in the form of non-clausal sequents rather than clauses, formulas or rewrite rules. The sequents are broken down toward conjunctive normal form, but existential quantifiers are not removed: we do not Skolemize. Avoiding Skolemization is one of the characteristics that define the IPR framework.

First we explain the widely used clausal form and give rules for transforming any formula into this form. Then we mention the disadvantages to storing knowledge in clausal form and the motivation for storing information in the non-clausal form. In Section 2.3.2, we present the rules for transforming a formula into the non-clausal form. It is the non-clausal form that is used by IPR and we will concentrate on this form in the bulk of this dissertation. We also present an illustrative example of a set of sequents being formed from an input definition. Finally, in Section 2.3.3, we prove certain soundness and completeness results that are used later.

2.3.1 Clausal Form

Here we present a novel presentation of a method for transforming a formula into clausal form. It is only novel in that it uses sequent notation and retains the ordinary semantics of sequents. When Fitting explains the rules for transforming a formula into clausal form he uses something like a tableau format but he reverses the roles of the α- and β-rules [54]. Smullyan showed a simple isomorphism between tableaux and trees of sequents [113]. Therefore, in order to make

the rules produce output that is easy to explain, we use the sequent form instead of the tableau form. Also, instead of reversing the roles of the α- and β-rules as Fitting does, we reverse the roles of the δ- and γ-rules. The reason for this is to maintain the semantics of sequents with which Fitting was not concerned in his presentation.

We stick as closely as possible to Smullyan's unified notation. Recall that in Smullyan's notation (see Chapter XI, Section 1 of *First-Order Logic* [113]) formulas in the antecedent of a sequent correspond to positive formulas on a branch of a tableau and formulas in the consequent of a sequent correspond to negative formulas on a branch of a tableau. Therefore, an existentially quantified formula in the consequent of a sequent is a γ-formula.

Recall that if Γ and Δ are sets of formulas, then a sequent, $\Gamma \rightarrow \Delta$, is thought of as asserting that the disjunction of the formulas in Δ logically follows from the conjunction of the formulas in Γ.

Suppose X is an axiom, definition or theorem of a theory expressed as a sentence. In order to transform X into clausal form, we build a sequent tableau rooted at the sequent $\rightarrow X$, that is, the sequent that asserts the truth of X. However, we do not use all of the ordinary sequent calculus rules.

Smullyan's rules for propositional connectives (α- and β-formulas) work fine in this context. For example, if we want the formula $A \supset (C \wedge B)$ to be in clausal form, we begin with the sequent $\rightarrow A \supset (C \wedge B)$ and apply the ordinary α- and β-rules from the sequent calculus so that the sequents $A \rightarrow C$ and $A \rightarrow B$ result. Notice that this is just the clausal or Kowalski form of the original formula.

However, the quantifier removal rules must be reversed.[9] Following the semantics of sequents, if the formula we want to have in clausal form is of the form $(\forall x)P(x)$, then we begin by asserting $\rightarrow (\forall x)P(x)$ (a δ-formula.) Since we are assuming this universal statement to be true, we want to replace the bound variable with a free variable. That is, we want to apply a

[9] When Fitting [54] describes clausifying, he takes the opposite approach so that the original formula is negated, propositional rules are reversed and the quantifier removal rules are left the same. Our approach is more appropriate to the semantics of sequents with which Fitting was not concerned in this context.

γ-rule and introduce a new free variable to a formula of this form.

In a similar way, if we want to clausify a formula of the form $(\exists x)P(x)$, we create the sequent $\rightarrow (\exists x)P(x)$, which contains only a γ-formula, and apply the δ-rule to it. This introduces a new Skolem function symbol.

Summarizing, in the context of building a sequent tableau during the construction of clausal form, in addition to the α- and β-rules (propositional rules), we apply the free variable γ-rule once to each δ-formula and eliminate the δ-formula from the new sequent. In order to avoid confusion with other rules, we call this the δ^{-1}-rule. Also, to each γ-formula, we apply the δ-rule and call this the γ^{-1}-rule.

Here are the rules in sequent format. We use the ordinary α- and β-rules from Section 2.2.2 but we do not use the δ- or γ-rules of that section. Instead, we use the rules below. The notation A^a_b represents the expression obtained by replacing every free occurrence of the variable a in the formula A with the term b.

Definition 2.12 (δ^{-1}-rule)

$$\frac{\Gamma \rightarrow \Delta, A^x_{x'}}{\Gamma \rightarrow \Delta, (\forall x)A} \qquad\qquad \frac{A^x_{x'}, \Gamma \rightarrow \Delta}{(\exists x)A, \Gamma \rightarrow \Delta}$$

where x' is a new variable.

Definition 2.13 (γ^{-1}-rule)

$$\frac{\Gamma \rightarrow \Delta, A^x_{f(\vec{y})}}{\Gamma \rightarrow \Delta, (\exists x)A} \qquad\qquad \frac{A^x_{f(\vec{y})}, \Gamma \rightarrow \Delta}{(\forall x)A, \Gamma \rightarrow \Delta}$$

where f is a new Skolem function symbol and $\vec{y}$ is the vector of free variables in the γ-formula.

When all possible α-, β-, γ^{-1}- and δ^{-1}-rules have been applied, we will say that the conjunction of the sequents on the leaves of the resulting sequent tree is the clausal form for the input formula.

Example 2.5 Now let us look at the details of the example of the definition of a continuous function. The procedure begins with the desired definitional formula in the conclusions of a

sequent. In other words, the sequent that asserts the truth of formula 2.9 is constructed.

$$\rightarrow \left\{ \begin{array}{c} (\forall f)(\forall S)(\forall T)(\text{continuous-from-to}(f,S,T) \leftrightarrow \\ (\text{function-from-to}(f,\text{top-to-class}(S),\text{top-to-class}(T)) \wedge \\ (\forall G)(\text{open-in}(G,T) \supset \text{open-in}(\text{the-set-inverse-image}(f,G),S)))) \end{array} \right\} \tag{2.10}$$

Since the only formula in the sequent is a δ-formula, we apply the δ^{-1}-rule (three times), followed by α- and β-rules to obtain the following three sequents.

$$\begin{array}{c} \text{continuous-from-to}(f,S,T) \rightarrow \\ \text{function-from-to}(f,\text{top-to-class}(S),\text{top-to-class}(T)) \end{array} \tag{2.11}$$

$$\begin{array}{c} \text{continuous-from-to}(f,S,T) \rightarrow \\ (\forall G)(\text{open-in}(G,T) \supset \text{open-in}(\text{the-set-inverse-image}(f,G),S)) \end{array} \tag{2.12}$$

$$\left\{ \begin{array}{c} \text{function-from-to}(f,\text{top-to-class}(S),\text{top-to-class}(T)), \\ (\forall G)(\text{open-in}(G,T) \supset \text{open-in}(\text{the-set-inverse-image}(f,G),S)) \end{array} \right\} \rightarrow \text{continuous-from-to}(f,S,T) \tag{2.13}$$

No more rules apply to sequent 2.11. Since there is a δ-formula in the conclusions of sequent 2.12, we apply the δ^{-1}-rule to it followed by an α-rule and obtain the following sequent.

$$\left\{ \begin{array}{c} \text{continuous-from-to}(f,S,T), \\ \text{open-in}(G,T) \end{array} \right\} \rightarrow \text{open-in}(\text{the-set-inverse-image}(f,G),S) \tag{2.14}$$

No more rules may be applied to this sequent.

Sequent 2.13 has a γ-formula in the hypotheses and so we apply the γ^{-1}-rule followed by a β-rule to obtain the following sequents.

$$\left\{ \begin{array}{c} \text{function-from-to}(f,\text{top-to-class}(S),\text{top-to-class}(T)), \\ \text{open-in}(\text{the-set-inverse-image}(f,G(f,S,T)),S) \end{array} \right\} \rightarrow \text{continuous-from-to}(f,S,T) \tag{2.15}$$

$$\begin{array}{c} \text{function-from-to}(f,\text{top-to-class}(S),\text{top-to-class}(T)) \rightarrow \\ \left\{ \begin{array}{c} \text{open-in}(G(f,S,T),T), \\ \text{continuous-from-to}(f,S,T) \end{array} \right\} \end{array} \tag{2.16}$$

Where G is a new Skolem function of three arguments. No more rules apply to sequents 2.15 or 2.16.

Therefore, the clausal form of the original formula in sequent 2.10 is the conjunction of clauses 2.11, 2.14, 2.15 and 2.16.

Motivations for Rejecting Clausal Form As we mentioned in Chapter 1, the framework presented in this dissertation was developed to support an automated theorem proving program that is intended to have certain properties. These desired properties influenced the development of the framework. The two important properties we wanted the program to have were the ability to prove theorems in mathematics using knowledge and that it be easy to use by someone who is not an expert in ATP techniques. Therefore, we want it to be very easy for the user to understand the contents of the knowledge base and to understand how knowledge is applied.

The advantages to clausal form have to do with the fact that procedures using this form can take advantage of certain restrictions without losing completeness. These restrictions improve the efficiency of the proof search. However, clausal form has certain disadvantages that conflict with the motivations behind the IPR project.

The two clauses (2.11 and 2.14) are perfectly readable and sensible.[10] However, the other two (2.15 and 2.16) don't make sense on their own, logically or cognitively. These two clauses are not logical consequences of the input formula 2.9. Therefore, if a mathematician asked to see the set of assumptions stored in the computer in clausal form, he or she would be very confused by the presence of these two falsifiable items. Not only are these two items false in most models of the input formula, but they don't make any sense to the reader. Part of this comprehension problem has to do with the fact that the clauses are not true but part has to do with the presence of Skolem terms.

These facts make a formula in clausal form much more difficult to understand and relate to the formulas that the user input. In Section 2.3.2 we present this example again using the new rules we introduce there.

This was the initial motivation for avoiding Skolemization and for the use of the non-clausal form described in Section 2.3.2. Since then, the author has found some other advantages (and disadvantages) to using non-clausal form.

[10] The reader who is unfamiliar with these predicates and functions can simply imagine that they are undefined. No knowledge of mathematical topology is assumed.

Portability. The presence of Skolem terms in a knowledge base adds difficulty to the proposition of combining knowledge bases because there is a possibility of clash between the symbols.

Readability. The advantage that most motivated the adoption of the form that knowledge takes in this framework is readability. Part of the problem non-experts have in reading formulas in clausal form is easily relieved by writing the clauses in Kowalski or sequent form. However, this does not take care of the more serious logical problem caused by Skolem functions.

Cognitive and Logical Relationships. A clause is not necessarily a logical consequence of the formula from which it was formed. As a result, clauses are not true (in models of the theory in question) and therefore their presence in a knowledge base is very confusing to an uninitiated user.

Efficiency. Since formulas are stored at a slightly higher level than clausal form, matching can occur earlier in the theorem proving process. Also, and for the same reason, the knowledge base is smaller than if formulas were taken all the way to clausal form.

Completeness. The main *disadvantage* to this framework is that when we apply the set-of-support strategy (the weak connection condition) to the theorem application (ε-) rule, completeness is lost. This is explained more in Section 3.3.

2.3.2 Rules for Transformation into Non-Clausal Form

Here we present the rules for breaking formulas into the non-clausal form used by the IPR system. The rules are the same as the rules for converting into conjunctive normal form (i.e., clausal form) with the exception that we do not Skolemize.

The only difference between the rules for non-clausal form and the rules for clausal form is the γ^{-1}-rule. In particular, if the top-level quantifier of a γ-formula can be "pushed inward" to obtain an equivalent formula, then that may be done. We call this the γ^{-1}-rule for non-clausal form although it has other names such as "anti prenexing."

Definition 2.14 When all possible α-, β-, γ^{-1}- and δ^{-1}-rules for non-clausal form have been applied, we will call the resulting sequent tree a *finished KB-tree*. If the root sequent is $\rightarrow \Phi$, then we say that the tree is a finished KB-tree *for* Φ.

The sequents on the leaves of the finished KB-tree are added to the knowledge base.

In the remainder of the dissertation, when we say that a sequent s is *in a knowledge base* we mean that it is a leaf of the finished KB-tree of some first-order formula. We say that such a sequent is in *knowledge base form*.

The following example of an application of the method illustrates its effectiveness at simulating some aspect of the way humans think of theorems.

Example 2.6 Let us look again at the details of the example of the definition of a continuous function. Suppose we want the definition of a continuous function to be in the knowledge base. See Section 5.2.2 to see exactly what the user needs to input into the IPR program to accomplish this. The program begins with the desired definitional formula and puts it into the conclusions of a sequent. In other words, the sequent that asserts the truth of formula 2.9 is constructed.

$$\rightarrow \left\{ \begin{array}{c} (\forall f)(\forall S)(\forall T)(\text{continuous-from-to}(f,S,T) \leftrightarrow \\ (\text{function-from-to}(f,\text{top-to-class}(S),\text{top-to-class}(T)) \wedge \\ (\forall G)(\text{open-in}(G,T) \supset \text{open-in}(\text{the-set-inverse-image}(f,G),S)))) \end{array} \right\} \tag{2.17}$$

Since the only formula in the sequent is a δ-formula, the program applies the δ^{-1}-rule (three times), followed by α- and β-rules to obtain the following three sequents.

$$\begin{array}{c} \text{continuous-from-to}(f,S,T) \rightarrow \\ \text{function-from-to}(f,\text{top-to-class}(S),\text{top-to-class}(T)) \end{array} \tag{2.18}$$

$$\begin{array}{c} \text{continuous-from-to}(f,S,T) \rightarrow \\ (\forall G)(\text{open-in}(G,T) \supset \text{open-in}(\text{the-set-inverse-image}(f,G),S)) \end{array} \tag{2.19}$$

$$\left\{ \begin{array}{c} \text{function-from-to}(f,\text{top-to-class}(S),\text{top-to-class}(T)), \\ (\forall G)(\text{open-in}(G,T) \supset \text{open-in}(\text{the-set-inverse-image}(f,G),S)) \end{array} \right\} \rightarrow \text{continuous-from-to}(f,S,T) \tag{2.20}$$

No more rules apply to sequent 2.18 so it is added to the knowledge base. Sequent 2.20 has a γ-formula in the hypotheses, however, no rule applies to this γ-formula since the quantifier cannot be pushed inward. Therefore, sequent 2.20 is also added to the knowledge base as it is.

Since there is a δ-formula in the conclusions of sequent 2.19, the program applies the δ^{-1}-rule to it followed by an α-rule and obtains the following sequent.

$$\left\{ \begin{array}{c} \text{continuous-from-to}(f, S, T), \\ \text{open-in}(G, T) \end{array} \right\} \rightarrow \text{open-in}(\text{the-set-inverse-image}(f, G), S) \qquad (2.21)$$

Since no more rules may be applied to this sequent, it is also added to the knowledge base.

Therefore, the finished KB-tree for the original formula in sequent 2.17 has three leaves (the sequents 2.18, 2.20 and 2.21) that are added to the knowledge base.

Notice that sequents 2.18 and 2.21 are exactly the first two clauses from Example 2.5 while sequent 2.20 encodes the information from the last two clauses in Example 2.5 in a more comprehensible, readable and logical way. In particular, sequent 2.20 is a logical consequence of the input formula and is quite sensible.

Also notice that these are just the three bits of information mentioned in Example 2.4

2.3.3 Soundness and Completeness

We want to be sure that we do not lose any information when we put formulas into the knowledge base. We also want to be sure that we do not add any new information or change the meaning of the formula in any way. To do this, we prove that this procedure for creating a knowledge base is both complete and sound. By completeness we mean that the conjunction of the leaves of a finished KB-tree logically imply the formula at the root. By soundness we mean that the original formula logically implies all of the sequents that are leaves of its finished KB-tree.

Theorem 2.15 (Soundness of the Knowledge Base) If a sequent s is a leaf of a finished KB-tree rooted at $\rightarrow \phi$ then every model of ϕ satisfies the universal closure of s. (We use the notation $\phi \models s$ to indicate this.)

Proof: Suppose s is a leaf of a finished KB-tree rooted at the sequent $\rightarrow \phi$. There is a sequence $\rightarrow \phi = s^0, s^1, \ldots, s^m = s$ of the sequents between s and the root of the tableau. We will prove $\phi \models s$ by induction on m.

The base case ($m = 0$) is trivial.

Now suppose that $\phi \models s^i$ and show that $\phi \models s^{i+1}$. Suppose $s^i = U \rightarrow V$ and $s^{i+1} = U' \rightarrow V'$. There are four cases to consider depending on the rule that was applied to construct s^{i+1} from s^i. The cases for γ^{-1}-, α- and β-rules are merely tedious and left to the reader.

δ^{-1}-rule: Suppose $(\exists x)A \in U$ and $U' = (U \cup \{A^x_{x'}\}) \setminus \{(\exists x)A\}$ where x' is a new variable. We will prove this case in detail and leave the case in which the δ-formula occurs in V to the reader.

Let λ be a grounding substitution for s^{i+1}. We will show that $\phi \models s^{i+1}\lambda$. We know that $\phi \models s^i\lambda$ by the induction hypothesis.

First suppose that $\phi \models \bigwedge_{u_i \in U} u_i\lambda$ and hence $\phi \models v\lambda$ for some $v \in V$. Clearly $V' = V$ so $\phi \models \bigvee_{v_i \in V'} v_i\lambda$ and hence $\phi \models s^{i+1}\lambda$.

Now suppose that for some $u \in U, \phi \not\models u\lambda$. If $u \neq (\exists x)A$, then $\phi \models s^{i+1}\lambda$ since $u \in U'$. Otherwise, $\phi \not\models (\exists x)A\lambda$. But this implies that $\phi \not\models A^x_{x'}\lambda$ where x' is a new variable. Hence $\phi \not\models \bigwedge_{u_i \in U'} u_i\lambda$ and so $\phi \models s^{i+1}\lambda$.

Q.E.D.

Theorem 2.16 (Completeness of the Knowledge Base) Suppose ϕ is a closed formula and S is the set of leaves of the finished KB-tree for ϕ. Then every model of the universal closure of S satisfies ϕ.

Proof: We prove by induction the stronger result that at every stage in the construction of a KB-tree, any model of (the conjunction of) the leaves satisfies the initial formula ϕ.

This is clearly true in the base case.

Now suppose that i steps have been applied in constructing a KB-tree for $i \geq 0$ and that at each of these steps any model of the universal closure of the set of leaves of the tree satisfied the formula at the root. This time, we will prove the case where a δ^{-1}-rule is applied

to a δ-formula in the conclusions of a sequent. We leave the case where a δ^{-1}-rule is applied to a δ-formula in the hypotheses and the case where an α-, β- or γ^{-1}-rule is applied to the reader.

Suppose S_i is the set of leaves after step i and step $i+1$ is the application of a δ^{-1}-rule to the formula $(\forall x)A$ in the conclusions of the sequent $s^i \in S_i$, where $s^i = U \to V$. Thus, $s^{i+1} = U \to A^x_{x'}, V \setminus \{(\forall x)A\}$ where x' is a new variable and $S_{i+1} = (S_i \cup \{s^{i+1}\}) \setminus \{s^i\}$. We must show that every model of the universal closure of S_{i+1} is a model for ϕ. It is enough to show that every model of the universal closure of s^{i+1} is a model of s^i. Suppose $\mathcal{M}$ is a model for the universal closure of s^{i+1}. If $\mathcal{M}$ satisfies the universal closure of $A^x_{x'}$ then it satisfies $(\forall x)A$ since x' is a variable not occurring in A. If $\mathcal{M}$ is not a model of $A^x_{x'}$, then $\mathcal{M}$ is a model for the universal closure of $U \to V \setminus \{(\forall x)A\}$ and hence for the universal closure of $U \to V$.

Q.E.D.

Chapter 3

Application of Knowledge

> The proposed method for doing logic always begins from scratch for each theorem. This is quite different from the type of proof we encounter in Euclid, where it is essential that later theorems are proved with the help of earlier ones. While this problem of selecting relevant earlier theorems to apply appears unimportant in the domain of logic as dealt with by the method to be described, it has to be faced at some stage, and the writer does not have a ready general solution of it.
>
> —Hao Wang [116]

Mathematicians select and apply lemmas from an enormous knowledge base. The fact that this is a difficult task even for mathematicians can be seen by the fact that they sometimes go off in the "wrong" direction when trying to prove a theorem and sometimes make several tries before they come up with a proof. The extreme complexity of this problem overwhelms existing automated methods. We have already discussed the difficulty of this problem in Section 1.1.

At this point we have described a formal language and the basic rules of inference used in this language. In Section 2.3 we described how sentences in this language are stored in the knowledge base.

It remains to explain how knowledge stored in this form (or in clausal form) is selected and used in search for the proof of a new sentence. The strategy for selecting a piece of knowledge from a knowledge base makes more sense with the understanding of how the knowledge is used. Therefore, we begin by illustrating the way knowledge is used in an example from topology. Then we introduce examples of the theorem application rules informally using

the sequent calculus. In Section 3.2, we give the formal details of the rule using the tableau calculus. The single ε-rule given in Section 3.2 subsumes all of the rules in Section 3.1. We also prove that the rule is sound and that there is a complete procedure associated with the rule. Finally, in Section 3.3, we describe some strategies and restrictions that the IPR program implements in order to use the rule more intelligently.

The strategies incorporate the intuition that the knowledge applied should finish the proof if possible. If it does not finish the proof, then it should not add more complexity to the proof. The knowledge applied should definitely be related to the theorem at hand.

Throughout this chapter, we concentrate on the non-clausal form. All of the methods apply equally to a library of information in clausal form.

3.1 Informal Description

First, examine an English proof that uses knowledge so that we can have examples of some of the ways knowledge can be applied in a proof. The reader who is very familiar with tableau methods might want to read Section 3.2 before reading this one because that section presents the technical details that this section describes loosely.

Example 3.1 Let us consider a simple example of the proof of a theorem taken from John Kelley's text, *General Topology* [77]. This is the first part of Theorem 19 in Chapter 5 of that text. Here is what we want to prove to be true assuming that the axioms and theorems given earlier in the text are true.

If a product space is locally compact then every coordinate space is locally compact.

To capture the context, we need to write it like this:[1]

If $\prod_A^\tau X$ *is locally compact then for every* a, X_a *is locally compact.*

[1] Here, $\prod_A^\tau X$ represents the product topology where X is a bijection from the index set A to a set of topologies. We use $\prod^\tau$ rather than $\prod$ to distinguish the topology from the underlying set. If you are not familiar with these terms, just try to follow how this sentence is proved to be a consequence of the assumptions. The proofs are completely syntactic and do not rely on the meanings of the formulas.

Proof: Suppose that $\prod_A^\tau X$ is locally compact and show that for every a, X_a is locally compact.
Since we know that $\prod_A^\tau X$ is locally compact and we are trying to show that X_a is locally compact we can apply **a statement on page 147 of Kelley**.
Now we only need to show that $\pi_a : \prod_A^\tau X \to X_a$ is open and onto and $\pi_a : \prod_A^\tau X \to X_a$ is continuous and onto.
1. Since we are trying to show that $\pi_a : \prod_A^\tau X \to X_a$ is open and onto we can apply **Theorem 3.2 in Kelley** which finishes that branch of the proof.
2. Since we are trying to show that $\pi_a : \prod_A^\tau X \to X_a$ is continuous and onto we can apply **a statement on page 90 of Kelley** which finishes that branch of the proof.

Figure 3.1: The proof for Example 3.1.

In Figure 3.1, we present a proof of the theorem above, expanded to show all of the details. In fact, the proof is the output of the computer program, IPR, that implements the methods presented in this paper. (See Section 5.2 for the details on how IPR produces English output.) This example of a proof demonstrates some of the ways known information may be used in the process of proving a new sentence.

In the proof, there are three references to earlier statements from Kelley's text. Here are the three statements needed for the proof, each labeled with the description we will use to refer to them in the proof.

a statement on page 147 of Kelley:

If S is a locally compact space and there is an open, continuous function from S onto T, then T is locally compact.

Theorem 3.2 in Kelley:

The projection function, π_a, is an open, onto function.

a statement on page 90 of Kelley:

The projection function, π_a, is continuous and onto.

Notice that the application of **a statement on page 147 of Kelley** splits the proof into two parts. Let us examine the reason this happened. The theorem being applied, **a statement on page 147 of Kelley**, had four basic parts. It would be stored, as described in Section 2.3, as follows:

- *If* $f : S \to T$ *is open and onto and*
- $f : S \to T$ *is continuous and onto and*
- S *is locally compact,*
- *then* T *is locally compact.*

The IPR program would display the theorem as follows:

Description: a statement on page 147 of Kelley
Suppose all of the following:
1. F is an open function from S onto T
2. F is a continuous function from S onto T
3. S is locally compact
Then T is locally compact

At the point in the proof when this theorem is used, we know that $\prod_A^\tau X$ is locally compact and we are trying to show that X_a is locally compact. According to the theorem, all that is needed is a continuous, open function from $\prod_A^\tau X$ onto X_a. Once an onto function is selected as a candidate—in this case, π_a—it must be proved to be both continuous and open. So, the reason the proof is split into two parts is that there are two hypotheses of the theorem being applied that are not known to be true at the current stage of the proof. So these two hypotheses must be proved separately. Fortunately, these two subgoals are finished immediately by the application of the other two theorems.

Clearly, the difficulty of the proof is not in its complexity, but in selecting the knowledge from a large base of knowledge. Typically, a student might apply the definition of locally compact and the definition of the product topology before thinking of this shorter proof. We discuss some strategies for automating the selection of knowledge from a large knowledge base in Section 3.3.

This example only shows a couple of the ways theorems can be used. In this section we present rules that implement these and other ways by which theorems can be applied. For

the moment, we will hide some of the technicalities. We present the ε-rule informally in several parts below in order to show how it relates to the way mathematicians use theorems and write proofs. The technical details are presented in Section 3.2.

These rules for applying theorems depend on two basic facts: (1) the theorems are stored as sequents as described in Section 2.3 and (the sequents may be in clausal or non-clausal form) (2) a tableau-related proof procedure is being used to prove the new sentence. In this section, we assume that the proof procedure being used is the sequent calculus, whereas when we give the technical details, we use the tableau calculus.

Suppose that the theorem T from the knowledge base, which is stored as the sequent $U \rightarrow V$, has been selected for use in proving S, a sequent in the proof tree written as $\Gamma \rightarrow \Delta$.

We try to match formulas in U with formulas in Γ and we try to match formulas in V with formulas in Δ. All of the formulas matched must agree on what their shared variables are replaced with.

In general, not all of the formulas in U will be unifiable with formulas from Γ; so also for V and Δ. However, if all of the formulas in U and V are unified with formulas from the sequent being proved, then the sequent is true in any model of the theorem from the knowledge base. This is the special case of the ε-rule that we will want to apply when possible since it immediately finishes a branch of the proof.

Theorem Application Rule 3.1 Suppose U and V are the sets of antecedents and consequents of a sequent in knowledge base form and $\Gamma \rightarrow \Delta$ is a leaf of sequent tree in the process of being expanded for finding a proof. If all of the formulas in U and V are unified with formulas in Γ and Δ respectively, then the sequent $\Gamma \rightarrow \Delta$ is true assuming $U \rightarrow V$ is true. (The substitution σ must be applied.[2])

The second and third theorem applications in the example above were instances of the application of this form of the theorem application rule. In those two cases, the branches of

[2] In this presentation, for the sake of simplicity, we have eliminated the details about when and to what the substitution σ must be applied. A complete discussion of this is given in Section 3.2.

the proof on which the theorems were applied were immediately finished by the application of the theorem. We will present the example again in its entirety using these theorem application rules at the end of this section and again after the formal definition of the ε-rule.

We are not always fortunate enough to have a theorem in the knowledge base that immediately finishes the branch of the proof (i.e., closes the sequent) we are working on.

In order to state other special cases of the ε-rule, we let U' stand for the subset of U that was successfully matched (unified, technically) with a subset Γ' of Γ. Let V' stand for the subset of V that was successfully matched (unified, technically) with a subset Δ' of Δ. Suppose that σ is the substitution (assignment of variables) that matches these formulas. Therefore, U' contains the hypotheses of the theorem, T, that are known to be true in the current sequent, S, and V' contains the conclusions of the theorem, T, that are known to be false in the current stage of the proof, S.

Let $M = V \setminus V'$ and $N = U \setminus U'$ so that M is the set of conclusions of the theorem, T, that are not successfully unified with goals of the sequent, S, being proved and N is the set of suppositions of the theorem, T, that are not successfully unified with hypotheses of the sequent, S, being proved.

The rules for applying the theorem, T, will make use only of the sets M, N and the substitution σ. So the information needed for the application of the theorem is readily available.

Sometimes, when proving a statement, a person goes through the following reasoning process.

> I know A and I am trying to conclude B. I know a true sentence that says that if I know A and C then I will know that B is true. So, if I only knew that C were true, then I would know that B were true.

This says, under our notation, that $S = A \to B$ and $T = \{A, C\} \to B$ and thus $M = \emptyset$ and $N = \{C\}$. Therefore, we only need to prove C. The following rule is applied in that situation.

Theorem Application Rule 3.2 Suppose M and N are the sets of formulas from the sequent in the knowledge base that are not successfully unified with formulas in the sequent being

proved as described above. If M is empty and $N = \{N_1\}$ is a singleton then the following rule may be applied.

$$\frac{\Gamma \to N_1, \Delta}{\Gamma \to \Delta}$$

To illustrate another case, humans frequently go through the following thought process when proving a theorem:

> I know A and I am trying to prove B. I have a theorem that says that if A is true and B is false then C must be true. So if C were false, then B would be true and I would be done. So I can assume that C is true.

Under our notation, this says that $S = A \to B$ and $T = A \to \{B, C\}$, thus $N = \emptyset$ and $M = \{C\}$. In this case, we add C to the hypotheses of S. The following rule applies in this case.

Theorem Application Rule 3.3 Suppose M and N are the sets of formulas from the sequent in the knowledge base that are not successfully unified with formulas in the sequent being proved as described above. If N is empty and $M = \{M_1\}$ is a singleton then the following rule may be applied.

$$\frac{M_1, \Gamma \to \Delta}{\Gamma \to \Delta}$$

Many times, humans apply theorems in more complicated ways than these. As in the example we gave above of the proof in topology, sometimes there are more than one formula that remain to be proved when we apply a theorem. This and any other more complicated theorem application is covered by the following rule that adds more than one sequent to the sequent tree.

Theorem Application Rule 3.4 Suppose $M = V$ and $N = U$ as described above.

$$\frac{M_1, \Gamma \to \Delta \mid \cdots \mid M_m, \Gamma \to \Delta \mid \Gamma \to \Delta, N_1 \mid \cdots \mid \Gamma \to N_n, \Delta}{\Gamma \to \Delta}$$

where $M = \{M_1, \ldots, M_m\}$ and $N = \{N_1, \ldots, N_n\}$.

It is this rule that was applied first in the proof of the example above. See Section 3.2 for all of the details of these rules.

Notice that this rule adds $n + m$ branches to the tree. In fact, in all of these rules the amount of branching caused by the application of the rule will be $n + m$: the sum of the cardinalities of N and M. In Section 3.3, when we talk about how to select a sequent from the knowledge base to apply at a particular point in the theorem proving process, we will use this measure, $n + m$, as a major indicator of the usefulness of a sequent in the knowledge base.

Example 3.2 Now let us reconsider how the previous example from topology would be handled in this semi-formalized framework. The framework is completely formalized in Section 3.2.

If $\prod_A^\tau X$ *is locally compact then for every* a, X_a *is locally compact.*

We will prove this in the knowledge base containing the following sequents that correspond to the three theorems needed in the proof given earlier.

$$\rightarrow \text{continuous-from-onto}(\pi_a, \prod_A^\tau X, X_a) \tag{3.1}$$

$$\rightarrow \text{open-from-onto}(\pi_a, \prod_A^\tau X, X_a) \tag{3.2}$$

$$\left\{ \begin{array}{c} \text{open-from-onto}(f, A, B), \\ \text{continuous-from-onto}(f, A, B), \\ \text{locally-compact}(A) \end{array} \right\} \rightarrow \text{locally-compact}(B) \tag{3.3}$$

Section 5.2.3 shows how these sequents are displayed in plain English by the IPR program and in that section, this example is explained in terms of the actual interaction with the program.

To prove the theorem, we begin by posting the following sequent, which is the statement of the theorem in first-order logic.

$$\rightarrow (\forall X)(\forall A)(\text{locally-compact}(\prod_A^\tau X) \supset (\forall a)\text{locally-compact}(X_a))$$

The ordinary sequent calculus rules are applied to this until we obtain the following sequent.

$$\text{locally-compact}(\prod_{A_0}^\tau X_0) \rightarrow \text{locally-compact}(X_{0\,a_0})$$

Here, X_0, A_0 and a_0 are Skolem constants. No more ordinary first-order logic rules may be applied so we must apply a theorem. We will apply the instance of sequent 3.3 obtained by applying the substitution $\sigma = \{\prod_{A_0}^{\tau} X_0/A, X_{0\,a_0}/B\}$. Applying Theorem Application Rule 3.4, we split the tableau into the following two branches.

$$\text{locally-compact}(\prod_{A_0}^{\tau} X_0) \rightarrow \left\{ \begin{array}{c} \text{locally-compact}(X_{0\,a_0}), \\ \text{continuous-from-onto}(f, \prod_{A_0}^{\tau} X_0, X_{0\,a_0}) \end{array} \right\} \tag{3.4}$$

$$\text{locally-compact}(\prod_{A_0}^{\tau} X_0) \rightarrow \left\{ \begin{array}{c} \text{locally-compact}(X_{0\,a_0}), \\ \text{open-from-onto}(f, \prod_{A_0}^{\tau} X_0, X_{0\,a_0}) \end{array} \right\} \tag{3.5}$$

Sequent 3.4 is closed by applying Theorem Application Rule 3.1 with the known sequent 3.1 and the substitution $\sigma = \{A_0/A, X_0/X, a_0/a, \pi_{a_0}/f\}$. Sequent 3.5 is closed by applying Theorem Application Rule 3.1 with the known sequent 3.2 and the same σ as was used on sequent 3.4. Both of these applications of the theorem application rule close the respective branches and the substitutions used are identical. Therefore the tableau is closed by the substitution $\sigma' = \{\pi_{a_0}/f\}$.

The question that naturally arises in the context of the automation of reasoning is how to select, not only the proper sequent from a large knowledge base, but how to select exactly the proper way to apply the selected sequent. We address this question in Section 3.3. The strategies we develop there make exactly the selection of the sequents we used in the common-sense proof above even from a relatively large knowledge base. Not only are the correct sequents selected, but they are applied by those strategies exactly as we have applied them here in our proof.

3.2 Technical Details

In this section, we present the formal details of the theorem application rule (ε-rule.) We then prove that the rule is sound and that there is a complete procedure using the rule.

The procedure we prove complete here is complete regardless of whether the knowledge base is made of clauses or the non-clausal form described in Section 2.3 [64]. In Section 3.3 we mention how completeness is lost if certain strategies are used in conjunction with the non-clausal form but is retained if a clausal form is used in the knowledge base.

The definition of a closed tableau does not presuppose any particular method for determining when a tableau is closed. Since the depth-first backtracking strategy is so widely used[3] and since the provisions of the theorem application (ε-) rule depend on the unification strategy being used, we discuss the three common unification strategies here briefly. It is important that we mention breadth-first unification since that is the method used by the current implementation of IPR. Section 5.1.1 goes into more detail on the topic of breadth-first unification.

3.2.1 Unification Strategies

For each of these strategies, there are certainly refinements that make it more efficient. For example, Beckert's mixed universal and rigid unification [18] or Oppacher and Suen's *condense* algorithm [94] can be incorporated into any of these strategies with some additional bookkeeping.

Depth-First Backtracking Unification. In this strategy, we use a limit, q, on the number of times the γ-rule may be applied to a single formula. An implementation of this strategy expands the first branch until a substitution is found that closes that branch. That substitution is applied across the entire tree (in a way so that it can be retracted later if necessary) and then the next branch is expanded. This process continues until the entire tree is closed or the q-limit is reached. If the q-limit is reached when no branches have been closed then the procedure halts with failure. Suppose the q-limit is reached and B is the last branch to be closed and σ is the substitution that closed it. Then the application of the substitution σ is undone and the branch B is expanded further until a different closing substitution is found or the q-limit is reached. When the procedure halts with failure, then we know that there is no proof of the formula with the given q-limit. At this point, the user may raise the q-limit and try again.

[3] IPR is widely believed to be the only tableau based prover that does not use depth-first unification. This comment was made by Ulrich Furbach at the most recent TABLEAUX conference.

Depth-First Non-Backtracking Unification. Our description will follow Frank Brown's description of the strategy [39]. With each γ-formula, we associate a *replica instance list*, which is the list of instantiations of the variables introduced by the application of a γ-rule to the formula. Each γ-formula is initialized with an empty replica instance list. On each application of a γ-rule to the γ-formula, the variable introduced by the application of the rule is put in the replica instance list of the γ-formula.

The first branch of the tree is expanded until a substitution is found that closes the branch and that does not violate the *instantiation restriction*. The *instantiation restriction* states that a variable may not be instantiated with a term that already occurs in the replica instance list of the γ-formula that introduced the variable. After the substitution is applied to the entire tree, the next branch is expanded. In this case, the substitution may be applied to the tree in a permanent way since it will never be undone. The substitution is even applied to the replica instance lists of the γ-formulas. Therefore, the replica instance lists are, as was mentioned, lists of instantiations of the variables introduced by the formula. This process continues until some resource is exhausted or every branch is closed.

Breadth-First Unification. The breadth-first strategy expands all branches of the tree. When a substitution is found that closes some branch, the substitution is stored at the highest node, N, such that every branch through N is closed by the substitution. Notice that a branch is expanded further even after a substitution is found that closes it. At certain times, the implementation must pass through the tree trying to find a composition of substitutions that will close every branch of the tree. No substitution is applied until a substitution is found that closes every branch.

The breadth-first method requires more bookkeeping but allows all strategies that are allowed by the depth-first search plus many other strategies, such as easy subgoal selection, dynamic lemma creation, etc.

3.2.2 The ε-Rule

Here we describe the technical details of a general and complete rule for applying sequents from a knowledge base in the context of a free-variable semantic tableau proof [54, 113]. We prove soundness and completeness and illustrate the use of the rule by using it to prove the theorem of topology we have been discussing.

If the ε-rule is used in the context of a clausal form knowledge base, then the rule is very similar to other clausal tableau expansion rules but it adds fewer branches to the tableau even in the clausal case. In that sense, S-tableaux (defined below) are similar to hyper-tableaux [15] and tableaux constructed by MGTP [66], which are clausal tableaux. However, the present calculus is more symmetric and has other differences discussed in Chapter 7. The ε-rule was developed in 1994 [106, 107], but since the author was not using clausal form, the relation between this work and the similar work done at that time and since that time in clausal tableaux was not immediately noticed. Further comparison is given in Chapter 7.

In the rules of the analytic tableau calculus, the symbol naming the rule (e.g., α) is also used to represent the formula to which the rule is to be applied. Since the ε-rule is applied to a *set* of formulas, we let ε stand for a (possibly empty) set of formulas on a branch of a tableau. Each element of ε must be either a γ-formula or a literal (if clausal form is being used, then ε will contain only literals.) The ε-rule is the application of a sequent in the knowledge base to the formulas in ε.

Let $s = U \rightarrow V$ be the selected sequent from the knowledge base. In order to apply the ε-rule, ε and s must satisfy the following conditions:

- s must have its free variables renamed so that they are disjoint from the free variables in the tableau,
- $\varepsilon = H \cup G$ where H is the set of positive formulas in ε and G is the set of negative formulas in ε and
- there are subsets U' and V' of U and V respectively and an idempotent substitution, σ,

such that[4]

- $H\sigma = U'\sigma$,
- $\{\psi\sigma | \neg\psi \in G\} = V'\sigma$,
- σ is a most general substitution with these properties.

For the statement of the ε-rule, we let $M = V\sigma \setminus V'\sigma$ and $N = U\sigma \setminus U'\sigma$. Loosely speaking, M contains exactly the formulas in V (the conclusions in s) that were not successfully unified with any formula in G (the negative formulas in ε) and similarly for N with respect to the hypotheses and positive formulas. We will also let σ' denote the subset of σ containing only those substitutions of variables in ε.

Definition 3.5 (ε-rule) Where $M = \{M_1, \ldots, M_m\}, N = \{N_1, \ldots, N_n\}$ and σ' have been determined as described above we apply the following tableau expansion rule to every branch containing ε.

$$\frac{\varepsilon}{M_1 \mid \cdots \mid M_m \mid \neg N_1 \mid \cdots \mid \neg N_n}$$

with the following provisions (for depth-first unification):

- none of the new formulas already occurs on the branch,
- the tableau substitution rule must be applied with σ' and
- if $M = N = \emptyset$ then the branch is closed by the application of the rule.

The provisions are stated assuming that depth-first unification is being used. We will discuss below how they need to be restated if breadth-first unification is being used. The first provision is not needed for soundness but it does not take away any first-order completeness and is generally useful. This is what is called the *regularity* condition [82]. The ε-rule would be unsound without the second provision.

[4] A substitution, σ, is idempotent if for any formula, F, $F\sigma = F\sigma\sigma$.

If breadth-first unification is being used, then the provisions change. It goes against the idea of breadth-first unification to apply σ' across the entire tree until it is known to be the substitution that finishes the proof. However, it is not enough simply to apply the substitution σ' to the formulas being introduced by the ε-rule because soundness is lost.

Here is a solution that works and maintains the goals of breadth-first unification: the substitution σ' need not be applied across the entire tree but it must be associated with the formulas added by the rule and their descendents. In subsequent applications of the ε-rule, the substitution formed must be composed with the substitutions associated with each of the formulas in ε and the composition of those substitutions must be associated with the formulas created by the ε-rule. Furthermore, if a contradiction is sought between two formulas on a branch, the substitution that unifies them must be composed with the substitutions associated with the two formulas.

We do not apply the first provision given in the ε-rule because it may destroy completeness if breadth-first unification is used since the two identical formulas may have different substitutions associated with them. Keeping track of regularity during breadth-first unification is an interesting topic for further research. Regarding the third provision, it should be understood that instead of closing a branch we label the appropriate node with σ'. Please see Section 5.1.1 for more details about breadth-first unification.

Example 3.3 Here we illustrate the use of the ε-rule in our running example. Here the proof will be given in tableau format.

If $\prod_A^\tau X$ *is locally compact then for every* a, X_a *is locally compact.*

We will prove this in the knowledge base containing the same sequents given in Example 3.2.

We construct the tableau in Figure 3.2 using the ordinary free-variable tableau rules until we reach node 2.

After node 2 has been added, no more ordinary tableau rules may be applied so we apply an ε-rule. The set ε will contain the only two literals in the tableau (1 and 2.) We

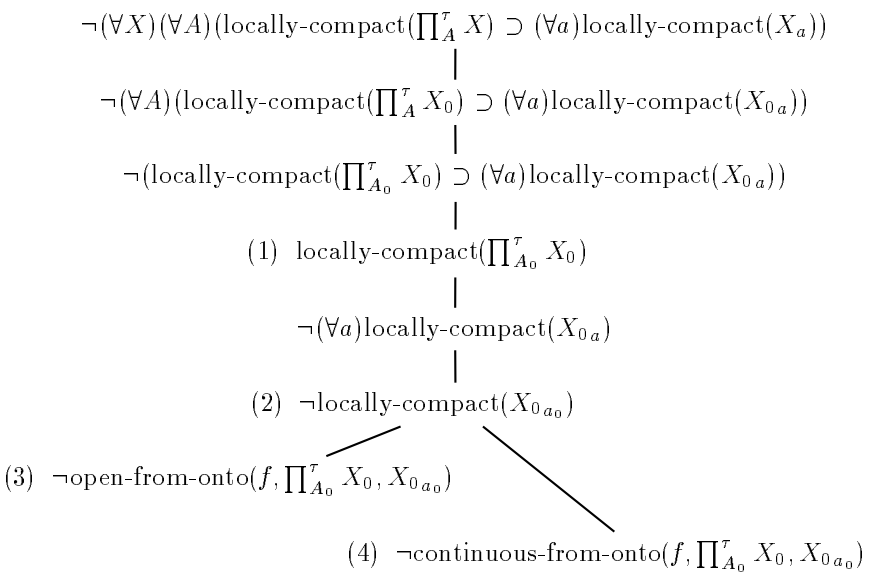

Figure 3.2: The tableau for Example 3.3. The symbols X_0, A_0 and a_0 are Skolem constants.

will apply the sequent 3.3 with the substitution $\sigma = \{\prod^{\tau}_{A_0} X_0/A, X_{0\,a_0}/B\}$. Applying the ε-rule, we split the tableau into two branches as shown. The left branch is closed by applying the ε-rule with ε containing only formula 3. We use the sequent 3.2 and the substitution $\sigma = \{A_0/A, X_0/X, a_0/a, \pi_{a_0}/f\}$. The right branch is closed by applying the ε-rule with ε containing formula 4. We use sequent 3.1 and the same σ as was used on the left branch. Both of these applications of the ε-rule close the respective branches and the substitutions used are identical. Therefore the tableau is closed by the substitution $\sigma' = \{\pi_{a_0}/f\}$.

This proof is found by the IPR implementation even in the presence of a knowledge base containing over 100 sequents related to the concepts of products, product topologies, (locally) compact spaces, etc. See Section 3.3 for a description of the strategies used by IPR.

Definition 3.6 (S-tableau) If S is the set of sequents from the knowledge base used in applications of the ε-rule in the construction of a tableau, then the tableau is called an *S-tableau*.

If we let S denote the set of sequents used in this proof, (i.e., sequents 3.1, 3.2 and 3.3) then our soundness result will show that since the S-tableau is closed, the theorem proved is true in every model of the formulas input to create the sequents in S.

3.2.3 Soundness and Completeness

In this section we prove that the theorem application rule (the ε-rule) is sound and that there is a complete procedure using the rule. That the rule is sound means that we can only prove sentences that are true in every model of the sentences input into the knowledge base. That a set of rules using the ε-rule is complete means that any sentence that is true in every model of the knowledge base can be proved using the rules.

Here is the formal statement of the soundness theorem.

Theorem 3.7 (Soundness) Suppose Φ is a finite set of first-order sentences and S is a subset of the leaves of the finished KB-trees for the sentences in Φ. If an S-tableau with root labeled $\neg\phi$ is closed, then ϕ is true in every model of Φ.

The proof will use the following lemma.

Lemma 3.8 Suppose Φ is a finite set of first-order sentences and S is a subset of the leaves of the finished KB-trees for the sentences in Φ. Further, suppose that T is an S-tableau whose root is labeled by a sentence ψ. If the universal closure of S is true under an interpretation, $\mathcal{M}$, and $\models_{\mathcal{M}} \psi$, then the universal closure of T is true under $\mathcal{M}$.

Proof: Let $S, \mathcal{M}$ and T be as in the hypotheses. Since the root of T is labeled by ψ, there is a sequence $\psi = T^0, T^1, \ldots, T^m = T$ of tableaux, where T^{i+1} is constructed from T^i by a single tableau expansion or substitution rule. By induction on m, we will prove that $\models_{\mathcal{M}} T^m$.

By hypothesis, $\models_{\mathcal{M}} \psi$, so the case $m = 0$ is finished.

Now suppose that every ground instance of T^i is true under $\mathcal{M}$. We will show that every ground instance of T^{i+1} is true under $\mathcal{M}$. The case for the soundness of the α-, β-, δ-, γ- and tableau substitution rules is made elsewhere [54, 20]. We will show that this holds in the case that the ε-rule was applied.

Let λ be a grounding substitution for T^{i+1}. We will show that $T^{i+1}\lambda$ is true under $\mathcal{M}$. By induction, any ground instance of T^i is true under $\mathcal{M}$. If σ is the substitution that

is applied to T^{i+1} in the application of the ε-rule, then $\sigma\lambda$ is a grounding substitution for T^i. Therefore, there is some branch, B^i in T^i, such that $\models_{\mathcal{M}} B^i\sigma\lambda$.

Let B^i be such a branch. First, suppose that $\varepsilon \not\subset B^i$. Then the ε-rule was not applied to the branch B^i. Thus B^i is a branch in T^{i+1} and since $B^i\lambda$ is true under $\mathcal{M}$, we know that $\models_{\mathcal{M}} T^{i+1}\lambda$.

Now suppose that $\varepsilon \subset B^i$. Let $\varepsilon = H \cup G$ where H is the set of positive formulas in ε and G is the set of negative formulas in ε. Suppose that $s \in S$ is the sequent used in this application of the ε-rule where $s = U \rightarrow V$. By the definition of the ε-rule, there are sets U', V' subsets of U and V respectively and a substitution σ such that $H\sigma = U'\sigma$ and $\{\psi | \neg\psi \in G\sigma\} = V'\sigma$. We also have the sets $M = V\sigma \setminus V'\sigma$ and $N = U\sigma \setminus U'\sigma$.

By hypothesis, the universal closure of s is true under $\mathcal{M}$, since $s \in S$. In particular, $\models_{\mathcal{M}} s\sigma\lambda$. Since $\varepsilon \subset B^i$ and $\models_{\mathcal{M}} B^i\sigma\lambda$, we know that $\models_{\mathcal{M}} \varepsilon\sigma\lambda$.

It will help to write $s\sigma\lambda$ as $H\sigma\lambda \cup N\lambda \rightarrow \{\varphi | \neg\varphi \in G\sigma\lambda\} \cup M\lambda$.

Since $\models_{\mathcal{M}} \varepsilon\sigma\lambda$ and $\models_{\mathcal{M}} s\sigma\lambda$, we know that $\models_{\mathcal{M}} \nu$ for some $\nu \in M\lambda \cup \{\neg f : f \in N\lambda\}$. This is enough because the branch, B^i, is extended to $n + m$ new branches each containing a unique one of the formulas in $M\lambda \cup \{\neg f : f \in N\lambda\}$. If $\mathcal{M}$ satisfies one of these formulas, then $\models_{\mathcal{M}} T^{i+1}\lambda$.

Q.E.D.

The proof of soundness will use the contrapositive of this lemma. In particular, we use the fact that if Φ is a finite set of first-order sentences and S is a subset of the leaves of the finished KB-trees for the sentences in Φ and T is an S-tableau whose root is labeled by a sentence ψ and T is not satisfiable, then no model of S satisfies ψ.

Proof of Soundness: We suppose Φ that is a finite set of first-order sentences and S is a subset of the leaves of the finished KB-trees for the sentences in Φ. Further suppose that T is a closed S-tableau with root labeled $\neg\phi$. We must show that ϕ is true in every model of Φ.

Since T is closed, T is not satisfiable. Therefore, $\neg\phi$ cannot be satisfied by any model of S by the contrapositive of Lemma 3.8. Thus every model of S satisfies ϕ. Since every model

of Φ satisfies S by Theorem 2.15, every model of Φ satisfies ϕ.

Q.E.D.

Now we present a simple systematic tableau procedure using the ε-rule that is complete for any first-order theory. The procedure given here is not recommended for implementation since it is very inefficient compared to other existing procedures. The IPR program does not use the procedure given here and, in fact, IPR is incomplete because it insists that $\varepsilon \neq \emptyset$.

The completeness proof is completely routine and follows the well-known proofs [54]. We will outline the ordinary proof using Hintikka's Lemma.

The procedure given below is complete regardless of whether clausal or non-clausal form is used. If clausal form is used, then certain restrictions such as set-of-support (Section 3.3) can be applied without losing completeness. The systematic procedure given here makes no effort to be efficient. See Section 3.3 for a description of a restriction to the rule that destroys completeness in the non-clausal case but gives good success.

Suppose we are given a finite set Φ of first-order sentences and that they have been used to create a knowledge base, S. That is, S is the set of leaves of the finished KB-trees for the formulas in Φ. Suppose that each sequent in the knowledge base, S, is labeled with a counter initialized to 0. To prove that ϕ is a logical consequence of Φ we build a systematic tableau following these steps. At step 0, we initialize the tableau with the formula $\neg\phi$ and we set $q = 1$. Every formula in the tableau has two labels on it: {used|unused} and a *copies* label that holds an integer. Each new formula starts out unused and with its copies set to 0. At step $n + 1$, if there is an unused α-, δ- or β-formula, then we apply the appropriate rule to the highest such formula and label the formula used. Otherwise, if there is a γ-formula whose copies label is less than q, then we apply the γ-rule to the highest such formula and increment the copies label of the γ-formula. Otherwise, if there is a sequent in the knowledge base, S, whose counter is less than q, then we apply the most general ε-rule for that sequent to every branch of the tableau on which it is permissible by the provisions of the ε-rule. Then we increment the counter of that sequent. If none of these rules can be applied, then we increment

q and continue. An S-tableau constructed by these rules is *finished* if it is infinite or if q is incremented twice without any other rule being applied.

Now we make the definition of the appropriate type of Hintikka set so that the proper form of Hintikka's Lemma will follow.

Definition 3.9 (Hintikka set for a knowledge base) Let L be a first-order language and S be the set of leaves of the finished KB-trees for the sentences (in the language) in the finite satisfiable set, Φ. A set H of formulas of L is a *Hintikka set for the knowledge base* S (with respect to L) provided H satisfies the following conditions:

H_0 No complimentary pair occurs in H

H_1 $\bot, \neg\top \notin H$

H_2 If $\alpha \in H$ then $\alpha_1, \alpha_2 \in H$

H_3 If $\beta \in H$ then $\beta_1 \in H$ or $\beta_2 \in H$

H_4 If $\gamma \in H$ then $\gamma(t) \in H$ for every closed term t of L

H_5 If $\delta \in H$ then $\delta(t) \in H$ for some closed term t of L

H_6 For every sequent $s \in S$ if $\vec{x}$ is the vector of free variables in s and

$$s = \{U_i(\vec{x}) | 1 \leq i \leq u\} \rightarrow \{V_i(\vec{x}) | 1 \leq i \leq v\}$$

then for every vector $\vec{t}$ of closed terms in L, some element of $\{\neg U_i(\vec{t}) | 1 \leq i \leq u\} \cup \{V_i(\vec{t}) | 1 \leq i \leq v\}$ is in H.

Lemma 3.10 (Hintikka's Lemma) If Φ is a finite satisfiable set of sentences (in a language with a non-empty set of closed terms) and S is the set of leaves of the finished KB-trees for the sentences in Φ, then every Hintikka set for S (with respect to the language) is satisfiable by some model of S.

The ordinary proof of Hintikka's Lemma goes through [54, 113]. First build an interpretation which makes the literals in H true, then show that this interpretation makes every other formula true by induction. The interpretation constructed in this way is easily seen to be a model for S because every ground instance of every sequent in S will be true in the model.

Theorem 3.11 (Completeness) If Φ is a set of first-order sentences and ϕ is a sentence that is true in any model of Φ, then the systematic procedure given above will find a proof of ϕ using the knowledge base built from Φ.

Proof: Since every model of S satisfies Φ by the completeness of the knowledge base (Theorem 2.16), every model of S satisfies ϕ. If the finished S-tableau for $\neg\phi$ has an open branch B, then we claim that there is a ground substitution, σ, such that $B\sigma$ is a Hintikka set for S. This must be so because of the fact that our procedure guarantees that there are infinitely many variants of the γ-formulas and the formulas from the sequents in the knowlege base and that the α-, β- and δ-rules will always be applied to the appropriate formulas. Therefore, σ can be constructed in a way that every term in the language will appear as the argument to every γ-formula and similarly for the formulas from the sequents.

Thus some model of S satisfies $B\sigma$ by Hintikka's Lemma. Because $\neg\phi \in B$ and $\neg\phi$ is a closed formula, this model satisfies $\neg\phi$. But this contradicts the fact that every model of S satisfies ϕ. Thus there can be no open branch on the finished S-tableau for $\neg\phi$. So the finished S-tableau for $\neg\phi$ must be closed.

Q.E.D.

3.3 Selection of Knowledge

> Therefore, one major (THE major) research activity is to devise methods (strategies) for guiding the program so that it makes enough of the right deductions and not too many of the wrong ones.
>
> —Bledsoe and Henschen [25]

In the foregoing sections, we have introduced a method for storing knowledge in a formal language and a method for using this knowledge in the process of establishing the truth of new sentences in the language. The method has been demonstrated to be quite compatible with human methods. (In Chapter 5 we give more examples showing how easy this method is to explain in human terms.) In this section, we discuss methods for selecting knowledge from a knowledge base for use in the theorem-proving process. The problem of selecting knowledge intelligently is very difficult. In a way, it is an impossible problem that even humans cannot solve perfectly.

We discussed in Chapter 1 the problems associated with selecting knowledge from a large knowledge base. But let us not give up before we begin. Indeed, computer systems have already been developed that use rather large knowledge bases and make selections automatically [11, 13, 62, 84, 123]. These systems show their strength in and, to a large extent, are limited to Horn theories and theories that have the nature of rewrite rules. Other systems exist that apply knowledge based on hints or other information from the user [36, 53, 61, 84]. There are also existing systems that use analogy: given a proof that is expected to be similar to the proof of the problem at hand, the prover is able to revise or expand the analogous proof [95].

So a problem that remains largely open is that of automatically discovering a proof in a large non-Horn theory when we have no idea what the proof is supposed to look like. It is in these circumstances that the application of knowledge can add a great deal of complexity to the proof. It is this seemingly impossible problem that mathematicians solve, sometimes with little conscious effort and sometimes with great difficulty.

We obviously want to apply knowledge that is related to the problem at hand. But how is this measured? And what happens when there is a tremendous amount of related knowledge? When trying to prove a theorem of mathematics, a mathematician prefers to use knowledge that is a bargain. He or she wants the statement to take the proof forward (or backward) as far as possible without paying for it with added complexity and without having to make a variable instantiation that he or she is not sure about.

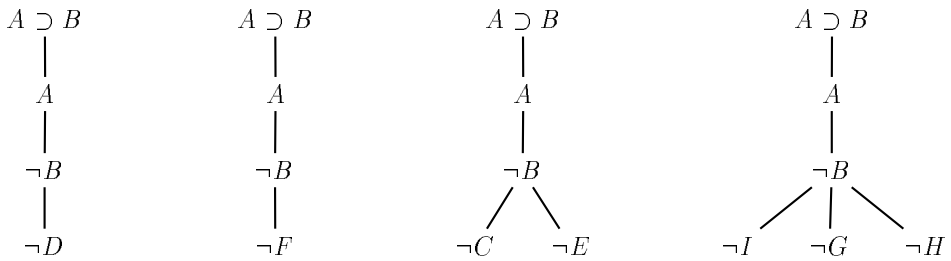

Figure 3.3: The tableaux for Example 3.4

Example 3.4 In order to get a naive idea of some of the heuristic decisions made by a human, consider a very abstract example. Suppose we want to prove that $A \supset B$ and we know, among other things, that each of the following are true.

1. $(A \wedge D) \supset B$,
2. $F \supset B$,
3. $(C \wedge E) \supset B$,
4. $(I \wedge G \wedge H) \supset B$,
5. B,
6. $\neg A$ and
7. $A \supset B$.

See Figure 3.3 to see the result of applying each of the first four lemmas using the ε-rule. Notice that after applying lemma 1 or 2, the tableau consists of a single branch. Lemma 3 causes a two-way branch and lemma 4 causes a three-way branch. Other things being equal, we would rather apply lemma 1 or 2 than lemma 3 because afterwards, we would only be required to prove D or F respectively whereas if we apply lemma 3, we would have to prove both C and E separately. This is the kind of complexity we want to avoid, if possible, because every branch is another case to consider. We would prefer to apply lemma 3 rather than lemma 4

since lemma 4 causes even more branching. It also seems that we would be more likely to use lemma 1 than lemma 2 in this case since lemma 1 appears to have more to do with what we are trying to prove than does lemma 2: lemma 1 has A and B in it whereas lemma 2 only mentions B.

Obviously we will prefer to apply one of the lemmas 5–7 (probably the last one) since any of these lemmas finishes the proof instantly.

Recall also the comments we have made about not wanting to make a variable instantiation while applying a theorem unless we have to. Summing up the simple observations we have made in this section, we obtain the following heuristics that seem sensible for selecting and applying lemmas:

1. Prefer to apply knowledge in a way that finishes the proof instantly.
2. Prefer to apply knowledge in a way that does not cause branching.
3. Prefer to apply knowledge in a way that causes as little branching as possible.
4. Prefer to apply knowledge in a way for which more of the parts in the statement match appropriately signed parts in the conjecture we are trying to prove.
5. Prefer to apply knowledge in a way that does not bind many variables.

Given the framework we have set up for storing and applying knowledge, it is very easy to implement these strategies. Each of these strategies can be reworded in terms of the information available to the theorem application (ε-) rule.

Items 1–3 are implemented simply by preferring to apply theorems for which $n + m$, the number of branches added by the theorem application, is small. This is the criterion that is given the most weight by IPR. It can be thought of as saying, "if applying a sequent this way causes the search space to grow too much, then don't apply the sequent that way." This "$n + m$" strategy, together with the framework for storing knowledge, seem to be major contributors to the success of IPR.

Item 4 is implemented by preferring to apply lemmas for which the sum of the cardinalities of U' and V' is large. Finally, item 5 is implemented by preferring to apply theorems in such a way that σ is small and not complex. We especially do not want too many variables from ε to be bound by the substitution σ'. This last strategy helps us keep our options open. We will find more ways to keep our options open when we study other aspects of the IPR implementation in Section 5.1.

The IPR program applies a restriction to the ε-rule that is not explicitly mentioned above.

Restriction 3.12 (Set-of-support) With the exception of the application of theorems involving equality (see Section 4.1.3), when a theorem is applied, $\varepsilon \neq \emptyset$.

That is to say that some formula from the theorem being applied must be unified with some formula in the tableau. (The handling of equality introduces a possible exception to this rule in IPR and is discussed in Section 4.1.) This restriction is called the weak connection condition in more recent work [64]. The set-of-support strategy was apparently first implemented in the context of tableau in the IPR prover and first described in an early technical report [106].

This restriction destroys completeness because of the fact that theorems are not stored in the form of clauses. However, examples that demonstrate this incompleteness are generally unnatural syntactic tricks.

Example 3.5 Suppose the axiom $(\exists x)(P(x) \wedge Q(x))$ is entered into the knowledge base. Without using an empty ε (i.e., the most general ε-rule), we can prove neither $(\exists x)(Q(x) \wedge P(x))$ nor $(\exists x)Q(x)$ from this knowledge base even though they are logical consequences.

However, by giving up a bit of completeness and resigning degenerate examples as these, a method may be found that is very successful on more natural examples. In addition, the framework facilitates human interaction.

It appears to follow from Hähnle's results that the ε-rule with this set-of-support restriction is complete in the context of a clausal knowledge base [64].

These are the strategies used by IPR in selecting lemmas. It is remarkable that the framework allows for such simple strategies to have impressive results (Chapter 6.) The kind of reasoning that IPR handles seems to be representative of a large area of reasoning that has been difficult for other methods. In Section 8.3 we discuss some limitations of the current method and how those can be overcome.

Chapter 4

Extensions to First-Order Logic

> We need also, *methods, procedures,* and *tricks of the trade*, which have been used so successfully by the great mathematicians over the years...
>
> —Woody Bledsoe [33]

It has been mentioned that first-order logic is an extremely expressive language and essentially all of mathematical or other reasoning can be framed in it. However, this is sometimes very tedious and inconvenient. Therefore, certain shortcuts are frequently used by mathematicians. In this chapter, we describe the implementation of some of these common shortcuts in a tableau based system. Except as indicated, the inference rules discussed here are implemented in the IPR prover.

The IPR program extends first-order logic to handle some equality reasoning and some higher-order (set theoretic) reasoning, which help it prove theorems in mathematics more easily and which make it easy to use.

The program IPR has shown its worth by proving many theorems independently. Simple examples illustrating the ideas are given throughout this chapter. Interesting examples of proofs using relatively large knowledge bases are provided in Chapter 6.

4.1 Equality

There are two basic ways of handling equality in a logic. One way is to include the axioms of equality in the knowledge and the second is to use special inference rules for equality. The first method is rarely used since the use of the axioms without special guidance introduces

tremendous complexity. Therefore, many special inference systems have been developed to handle equality [51].

IPR uses a combination of equality inference rules. The combination of the equality techniques used by IPR does not make IPR complete for equality problems. However, the rules presented here have proved to be extremely useful and powerful in practice.

By interacting, the user can force IPR to apply any equality from the knowledge base or from the hypotheses in various ways. Therefore, the user can use IPR to prove any theorem in an equality theory. In the remainder of this section, we describe only the equality techniques that IPR applies automatically.

4.1.1 Brown's Rule

Brown's Rule is a restricted equality substitution method [39].

Inference Rule 4.1 (Brown's Rule) If a branch contains a formula of the form $s = t$, where one side of the equality (say s) is a Skolem term that does not occur in the term on the other side (t), then add to the branch a copy of every unused formula on the branch containing s and replace all occurrences of s in the new formulas with t.

Brown's Rule alone has proved to be sufficiently powerful for most of the problems in Chapter 6. Apparently, eliminating Skolem terms when possible is all of the equality reasoning that is needed for a large class of problems in mathematics. However, problems in theories with an algebraic flavor will need more powerful methods.

A more liberal, yet still tractable restricted equality substitution rule for proving theorems in set theory is discussed in Section 4.2.4. The method mentioned there has not yet been implemented in IPR but has proved useful in examples proved by hand.

4.1.2 Congruence Closure

At an early stage in the development of IPR, it also used an incomplete E-unification algorithm incorporating the ground congruence closure technique [85, 92, 105]. This part of the code has not been kept current.

To use this technique, a grammar is built from the positive equalities on a branch and the simple equalities in the knowledge base. That grammar is used any time unification is performed between clashing terms.

The congruence closure method can be extended to handle class- and ι-terms. (See Section 4.2.) That is to say that to check for equality between such terms, the formulas in the terms need merely be alpha-congruent. If desired, this can be strengthened by equating two class expressions if the formulas in them are known (by a quick look-up method) to be equivalent in the branch.

In general, the congruence closure technique gives IPR power for more algebraic proofs. The congruence closure technique proves the following formulas easily. In each case, two tableau expansion rules (δ and α) are applied and then the congruence closure technique closes the tableau.

$$(\forall x)(x = f(x) \supset f(x) = f(f(f(f(f(x))))))$$

$$(\forall x)(x = f(x) \supset x = f(f(f(f(x)))))$$

With a little interaction, IPR can prove the following example as well. The user must manually tell IPR to make the substitutions of a and b for x.

$$(\forall a)(\forall b)(\forall c)((\forall x)(g(f(x)) = c \wedge f(x) = g(x)) \supset g(g(a)) = g(f(b)))$$

4.1.3 Miscellaneous Equality Rules

If a formula $s \neq t$ occurs on a branch of a tableau, then IPR tries to unify s with t. If it succeeds, then that substitution can be used to close the branch. (See Section 5.1.1 for the handling of breadth-first unification.)

IPR also applies equality reasoning when it is searching for applications of the ε-rule. In particular, if a formula $s = t$ is in N, the set of suppositions of the theorem that are going to be added negatively to the tree, then IPR tries to unify s and t in a way compatible with the substitution, σ, constructed already. If it is successful, then $s = t$ is removed from N and the substitution that unified s and t is composed into σ. This is the only case in which it is possible for IPR to apply a theorem even though $\varepsilon = \emptyset$.

Description: a basic fact about products.
Suppose
$x \in S \times T$
Then
for some a and b, $a \in S$ and $b \in T$ and $x = \langle a, b \rangle$

Description: a basic fact about products.
Suppose
1. $a \in S$
2. $b \in T$
3. $x = \langle a, b \rangle$
Then
$x \in S \times T$

Proof: Since we know that a is a member of S and b is a member of T and we are trying to show that $\langle a, b \rangle$ is a member of the product of S and T we can apply a basic fact about products which finishes that branch of the proof.

Figure 4.1: The knowledge base and proof excerpt for Example 4.1.

When a formula is eliminated from N in this way and the ε-rule is applied, we call this an application of the $\varepsilon_{=}$-rule.

The purpose of applying this rule is to keep the tableau small and narrow the search space. By removing a formula from N, $n + m$ decreases, which makes the sequent application more attractive.

Example 4.1 Here is a very simple example intended only to show how the $\varepsilon_{=}$-rule works.

Suppose the following is loaded into the knowledge base.

$$x \in S \times T \leftrightarrow (\exists a)(\exists b)(a \in S \wedge b \in T \wedge x = \langle a, b \rangle)$$

This is the definition of the cross product of two sets and it produces in the knowledge base the sequents in Figure 4.1. (They are displayed as output by IPR.) It will be given the label "a basic fact about products."

The following theorem is proved completely automatically with the knowledge of the theorem above.

$$(a \in S \wedge b \in T) \supset \langle a, b \rangle \in S \times T$$

$$\neg(\forall a)(\forall b)(\forall S)(\forall T)((a \in S \wedge b \in T) \supset \langle a, b\rangle \in S \times T)$$
$$|$$
$$\neg((a \in S \wedge b \in T) \supset \langle a, b\rangle \in S \times T)$$
$$|$$
$$a \in S$$
$$|$$
$$b \in T$$
$$|$$
$$\langle a, b\rangle \notin S \times T$$

$$(A)$$

$$\neg(\forall a)(\forall b)(\forall S)(\forall T)((a \in S \wedge b \in T) \supset \langle a, b\rangle \in S \times T)$$
$$|$$
$$\neg((a \in S \wedge b \in T) \supset \langle a, b\rangle \in S \times T)$$
$$|$$
$$a \in S$$
$$|$$
$$b \in T$$
$$|$$
$$\langle a, b\rangle \notin S \times T$$
$$|$$
$$x \neq \langle a, b\rangle$$
$$*$$
$$\{\langle a, b\rangle / x\}$$

$$(B)$$

Figure 4.2: The tableaux for Example 4.1. The symbols a, b, S and T are constants.

This proof requires some equality reasoning. Either of the methods mentioned above will suffice. We show a proof in Figure 4.2(A) that uses the $\varepsilon_{=}$-rule and in Figure 4.2(B) a proof that merely unifies terms in negative equality formulas.

In Figure 4.2(A), the proof is finished without adding the formula $x \neq \langle a, b\rangle$ because those two terms are unified during the application of the $\varepsilon_{=}$-rule. In Figure 4.2(B), the extra formula is added to the tableau and later the terms are unified. Both tableaux are closed. The first is closed by the $\varepsilon_{=}$-rule with $\sigma' = \emptyset$. The second tableau is closed by the indicated substitution.

In this very simple case, the use of the $\varepsilon_{=}$-rule made very little difference. However,

recall the importance of the $n+m$ strategy. In more interesting examples, the use of this shortcut can eliminate entire subtrees from the tableau. This occurs, for instance, in Example 6.1. If equality reasoning does not occur during the application of the ε-rule in that example, then IPR is unable to find the proof in a reasonable amount of time because the proof tree becomes too complex.

Figure 4.1 also contains an excerpt of the proof as it is explained by IPR. Notice that the application of the ε-rule finishes the proof easily and in a way that is easy to explain.

4.2 Comprehension and Description in Tableaux

> An inference rule that "builds in" set theory at the inference level is the objective of Research Problem 8. More precisely, just as the employment of paramodulation permits one to avoid using any equality axioms other than reflexivity, the sought-after inference rule for set theory would permit one to avoid using a number of the axioms in Gödel's approach.
>
> —Larry Wos [120]

Various approaches have been invented for enabling an automated theorem proving program to find proofs in set theory. Some take a small fragment of set theory and develop decision procedures [43]. Others prove theorems using a variant of Gödel's axioms of set theory [101]. Some invent special rules for handling properties such as the transitivity of the $\subset$-relation [69]. In proof-checking systems, of course, there is no limit to what a patient person with a great deal of expertise at the system can prove [96]. Probably the most closely related work to that presented here is the work of Frank Brown [39].

We build a strong but safe comprehension schema into the tableau calculus. The rules developed here are consistent with the comprehension schema of Kelley [77] or Bernays [21]. The method is compatible with various axiomatizations of set theory and is also useful in reasoning in more advanced theories in which the particular axiomatization of set theory is unimportant. The present approach is completely automatic and quite successful on many problems that are showcased as challenge problems for provers in set theory. In fact, this

procedure finds proofs of several of these examples without search. The method proposed here is also compatible with English explanation [110].

We implement the comprehension schema by means of tableau reduction (i.e., branch closure) and expansion rules. We also discuss the implementation of the definite descriptor in tableaux and special rules for handling equality effectively and in a tractable way in set theory.

The calculus described here is a step in an unusual direction toward a solution to Wos' eighth basic research problem—an inference rule for set theory [120]. It will be clear that the direction is very promising and that there is plenty of room for more work in this direction. It might be said that the rules here solve the problem in one direction (the easy direction.) That is, classes are easily deconstructed. The construction of classes is a much harder problem. Nevertheless, it will be shown that deconstruction is very powerful.

The focus of this work has not been completeness but efficiency and strength. A set of inference rules is presented for a subset of set theory and equality that is effective yet minimal (in a non-technical sense.) This results in a remarkably tractable and successful set of inference rules.

The comprehension schema is implemented by new tableau reduction and expansion rules. These rules were first described in an early technical report on the IPR prover [106] that implements the comprehension schema described here. Using these rules, the IPR system has proved many examples. The use of these rules allows the IPR user to express many commonly-used mathematical constructs much more easily than is possible with first-order logic. IPR also incorporates the ι-operator and has proved theorems involving it.

Some of the calculus described here has not yet been implemented in IPR. Unless otherwise indicated, all of the rules in Sections 4.2.1 and 4.2.2 are currently implemented. In those sections, we give many examples of theorems that IPR has proved using those rules.

None of the equality rules presented in Section 4.2.4 are currently implemented, except those that we described in Section 4.1. In Section 4.2.6, we describe some examples of theorems and discuss their proofs that can be found using the comprehension schema (Section 4.2.1) and the equality rules from Section 4.2.4.

4.2.1 The Comprehension Schema in Tableaux

The predicates handled by the built-in mechanism are $t \in C$, an-element(t) and a-class(C). The formula $t \in C$ is interpreted as meaning that t is "small" (i.e., an-element(t)) and C is a class (i.e., a-class(C)) and that t is an element of the class C. The formula an-element(t) means that t is either a (small) set or an ur-element. The formula a-class(C) means that C is a (possibly large) class.

The term constructor handled by the mechanism is the *classifier*, $\{y : A\}$, where A is a formula and y is a variable bound by the term. The term $\{y : A\}$ represents the class of all "small" objects, t, that satisfy the predicate A when every free occurrence of y in A is replaced by t. We will use the notation A_t^y to represent the formula that results from replacing every free occurrence of y in A by t.

IPR implements a version of the *comprehension schema* that can be stated as follows:

If y is a variable and A is a formula, then $\{y : A\}$ is a class. Further, if t is a term that is known to be a set or an ur-element then $t \in \{y : A\}$ if and only if A_t^y.[1]

Below we give the most general form of this schema presented here in tableau format.

$$\frac{t \in C}{\text{an-element}(t)} \qquad \frac{t \in \{y : A\}}{A_t^y} \qquad \frac{t \notin \{y : A\}}{\neg\text{an-element}(t) \mid \neg A_t^y}$$

where t and C are terms, y is a variable and A is a formula. Any branch containing a formula of the form

$$\neg\text{a-class}(\{y : A\})$$

is closed.

Another possible rule, which is not currently used in the IPR system, is the following.

$$\frac{t \in C}{\text{a-class}(C)}$$

[1] We often fail to mention the fact that no free variable in t can be bound in A since these problems are easy to handle by variable renaming, by keeping the sets of free and bound variables disjoint (as in Bernays [21]) or by some other method (as in Quine [103].) The reader who wants to implement this rule should be warned that this point is necessary for soundness.

This rule could also be stated as a branch closure rule. That is, any branch containing the formulas

$$t \in C$$
$$\neg\text{a-class}(C)$$

is closed. Which of these two rules is used by any particular implementation of this calculus depends on the control and other strategies used.

In a similar way, the first comprehension tableau expansion rule could be implemented as a branch closure rule: any branch containing a pair of formulas of the form

$$t \in C$$
$$\neg\text{an-element}(t)$$

is closed.

The following rules, which are simple combinations of the previously mentioned expansion rules and closure rules, might help to keep the search space small.

$$\frac{\text{an-element}(t) \quad t \notin \{y : A\}}{\neg A_t^y} \qquad\qquad \frac{t \in C \quad t \notin \{y : A\}}{\neg A_t^y}$$

These rules are redundant but may be useful in practice depending on the other strategies combined with the method. See Example 4.5.

Baaz, Egly and Fermüller have illustrated the usefulness of tableaux in the context of applying axiom schemata by applying the idea to induction and similar schemata [7]. In their paper, they argue the advantages of analytic or on-the-fly methods such as tableaux over preprocessed normal-form methods such as resolution. The same argument applies here. Using analytic tableau expansion and reduction rules to implement axiom schemata turns out to be very efficient and to have other advantages over methods that use preprocessing techniques such as resolution.

Other rules could be wired into a system [39] but those given here allow proofs to be found for an important and useful class of theorems. In the case of the comprehension schema, it is easy to see that other rules related to the ones described here may also be useful for some problems. For example, some rules related to the empty set might be useful. In Section 4.2.7

we mention some related work that involved this kind of thing. Except as specified, only the rules implemented in IPR are presented in this section.

Example 4.2 Here are some examples of set-theoretic facts that are proved without using any extra knowledge. That is to say, these are proved by IPR with an empty knowledge base. The proofs only use the comprehension rules that are wired into the prover.

Given a procedure for proving theorems and a theorem, we say that the procedure proves the theorem *without search* if the procedure proves the theorem with no waste and if no decisions are made in the proof. This definition is not very formal but the examples should make the meaning clear. Except as indicated, the proofs of these examples are found by IPR without search.

The only equality reasoning needed for these examples is the unification of the arguments of formulas of the form $s \neq t$.

This first example says that if a is small, then a is in the pair $\{a, b\}$.

$$\text{an-element}(a) \supset a \in \{x : x = a \vee x = b\}$$

$$\text{an-element}(c) \supset (c \in \{x : U(c)\} \leftrightarrow U(c))$$

$$\neg(\exists z)(z \in \{a : x = a\}) \leftrightarrow \neg\text{an-element}(x)$$

The next example says that $z \in X \cup Y$ if and only if $z \in X$ or $z \in Y$.

$$z \in \{a : a \in X \vee a \in Y\} \leftrightarrow z \in X \vee z \in Y$$

The next example says that the union of the power sets of X and Y is a subset of the power set of the union of X and Y.

$$\begin{array}{c} t \in \{a : a \in \{s : (\forall b)(b \in s \supset b \in X)\} \vee \\ a \in \{s : (\forall b)(b \in s \supset b \in Y)\}\} \supset \\ t \in \{a : (\forall b)(b \in a \supset b \in \{s : s \in X \vee s \in Y\}\} \end{array}$$

This example is also proved without search using only the comprehension rules. We discuss in later examples how this is proved in its more ordinary forms such as $\text{Pow}(X) \cup \text{Pow}(Y) \subset \text{Pow}(X \cup Y)$ and what we mean by "without search."

A stronger version of the dual is also proved easily using only the comprehension rules.

$$t \in \{a : (\forall b)(b \in a \supset b \in \{s : s \in X \wedge s \in Y\}\} \leftrightarrow$$
$$t \in \{a : a \in \{s : (\forall b)(b \in s \supset b \in X)\} \wedge a \in \{s : (\forall b)(b \in s \supset b \in Y)\}\}$$

This says that $\mathrm{Pow}(X \cap Y) = \mathrm{Pow}(X) \cap \mathrm{Pow}(Y)$ [32]. This is the only example so far that requires search. This proof requires search due to the fact that more than one variant must be made from a single γ-formula.

4.2.2 Description in Tableaux

Another common term constructor in set theory is the "definite descriptor" or the "ι-operator." The term $\iota_x(A)$ represents the unique small object that satisfies A if such a thing exists. Here x is a variable bound in the term and A is a formula. If there is no unique small object satisfying A, then $\iota_x(A) =$undefined. Here "undefined" is a special non-logical term in the language.[2]

In order to handle this term, we add one tableau expansion rule. Any formula P on a branch containing a term $\iota_x(A)$ is replaced by the formula[3]

$$\begin{array}{l}\text{if } (\exists r)(A^x_r \wedge \text{ an-element}(r) \wedge (\forall n)((A^x_n \wedge \text{ an-element}(n)) \supset r = n)) \\ \text{then } (\forall c)((A^x_c \wedge \text{ an-element}(c)) \supset P^{\iota_x(A)}_c) \\ \text{else } P^{\iota_x(A)}_{\text{undefined}}\end{array}$$

The user of the IPR system can add axioms specifying more information about "undefined." These axioms will be applied automatically using the ε-rule.

Example 4.3 Figure 4.3 contains some examples of theorems that are proved without using any extra knowledge. That is to say, these are proved by IPR with an empty knowledge base. The proofs only use rules that are wired into the prover. Each of these theorems requires the use of the ι-removal rule.

4.2.3 Axioms of Set Theory

Notice that these schemata do not bind the user to a particular choice of axioms of set theory. The rules are most useful and easy to use in a set theory like Kelley's [77] or Bernays' [21]. The author recommends the use of such a set of axioms for use in problems in low-level set theory.

[2] There are other ways of handling the case when the ι-term is undefined [21].
[3] The "if...then...else" logical operator can be handled by β-rules [99].

$$(\text{an-element}(b) \wedge a = \iota_x(x = b)) \supset a = b$$

$$\text{an-element}(a) \supset \iota_x(x = a) = a$$

$$\neg\text{an-element}(b) \supset \iota_s(s = b) = \text{undefined}$$

$$(\text{an-element}(c) \wedge U(c) \wedge (\forall r)(U(r) \supset r = c)) \supset c = \iota_r(U(r))$$

$$(\exists r)(\text{an-element}(r) \wedge U(r) \wedge (\forall n)((\text{an-element}(n) \wedge U(n)) \supset r = n)) \vee \iota_c(U(c)) = \text{undefined}$$

$$\neg(\exists x)U(x) \supset \iota_y(U(y) = \text{undefined}$$

$$(\text{an-element}(a) \wedge \text{an-element}(b) \wedge U(a) \wedge U(b) \wedge a \neq b) \supset \iota_r(U(r)) = \text{undefined}$$

$$(\text{an-element}(c) \wedge U(c) \wedge (\forall r)((\text{an-element}(r) \wedge U(r)) \supset r = c)) \supset U(\iota_x(U(x)))$$

$$(\forall r)(U(r) \leftrightarrow B(r)) \supset \iota_x(U(x)) = \iota_y(B(y))$$

Figure 4.3: Some examples of theorems involving ι-terms.

The comprehension schema bypasses many of Gödel's axioms because Gödel's set theory is a first-order theory [60]. (The idea of bypassing the axioms is suggested by Wos as the better way to solve his eighth research problem [120].) If the user is proving theorems in Gödel's theory, then the comprehension schema will never be applied since there is no classifier in Gödel's set theory. To prove theorems in Gödel's theory, an ordinary (or special purpose) first-order logic prover should be used.

However, in a proof system using the present rules, the user may make various decisions about the axioms to be used.

The user may choose to allow ur-elements or not. To disallow ur-elements, the user only needs to put the axioms

$$(\forall y)\text{a-class}(y)$$

and

$$(\forall x)(x = \emptyset \vee (\exists y)y \in x)$$

into the knowledge base. To disallow proper classes, the user only needs to put the axiom

$$(\forall y)\text{an-element}(y)$$

into the knowledge base. Disallowing proper classes in this way is very dangerous for consistency. In such a case, the user must take precautions about when the comprehension schema is applied. The user might assure that all of the formulas used in such a rule are stratified as described by Quine in his "new foundations" [103]. Or the user might be sure that each application of the rule is an instance of Zermelo and Fraenkel's axiom of subsets [21]. Of course, it is the duty of the user who puts axioms into the knowledge base to take care of consistency concerns.

The symbol, $\emptyset$, is a special non-logical constant in the language interpreted as the empty set.

The user may also be unspecific about the low-level axioms and work in an advanced mathematical theory without worrying about the low-level axioms.

4.2.4 Handling Equality in Set Theory

There are many theorems of set theory for which the comprehension schema described above is sufficient to find a proof. However, if some equality reasoning can be handled without adding too much complexity to the inference system, then more theorems can be proved and even those theorems that were proved without equality reasoning can then be proved in a more natural way. However, including a complete equality procedure can cause several problems in this context.

First, in this calculus, some kind of substitution of equals is necessary. For example, suppose we have the formulas $x = \{y : A\}$ and $a \in x$ on a branch. Then the comprehension schema cannot be applied unless the substitution of $\{y : A\}$ for x is made in the second formula. However, implementing a complete procedure that makes substitutions can be very inefficient [18].

Therefore, we introduce restricted equality substitution and extensionality rules here that are not terribly expensive to apply in realistic examples and that give more strength to

a prover. The idea is to have a set of rules that apply equalities on-the-fly just when they are needed. This helps an interface by keeping terms written in a nice abbreviated form until the definitions are needed.

Before giving the restricted substitution rule, we describe a rule that incorporates some of the strength of the axiom of extensionality.

These rules have not yet been implemented in IPR but they have proved successful and useful in examples done by hand.

Restricted Extensionality Rule The restricted extensionality rule applies only in the case of an equality (or the negation of an equality) occurring on a branch such that both arguments are class terms. These rules are nothing more than shortcuts to the use of the first-order axiom of extensionality in this special case. This allows for the full axiom of extensionality to be absent from the knowledge base while still allowing it to be applied in the special cases described here. The advantage to this is that applying the full axiom of extensionality in other cases can add a great deal of complexity to the proof search.

$$\frac{\{x : A\} = \{x : B\}}{\begin{array}{c|c|c} \text{an-element}(x') & \text{an-element}(x') & \neg\text{an-element}(x') \\ A^x_{x'} & \neg A^x_{x'} & \\ B^x_{x'} & \neg B^x_{x'} & \end{array}}$$

$$\frac{\{x : A\} \neq \{x : B\}}{\begin{array}{c|c} \text{an-element}(f(\vec{x})) & \text{an-element}(f(\vec{x})) \\ \neg A^x_{f(\vec{x})} & A^x_{f(\vec{x})} \\ B^x_{f(\vec{x})} & \neg B^x_{f(\vec{x})} \end{array}}$$

Here x' is a new free variable and $f(\vec{x})$ is a Skolem term where $\vec{x}$ is the vector of free variables occurring in the formula $\{x : A\} \neq \{x : B\}$.

A similar restricted extensionality rule can be written for the subset predicate, $\subset$. This rule also applies only in the case of a subset predicate (or the negation of a subset predicate)

occurring on a branch such that both arguments are class terms.

$$\frac{\{x : A\} \subset \{x : B\}}{\begin{array}{c|c|c} \text{an-element}(x') & \text{an-element}(x') & \neg\text{an-element}(x') \\ B^x_{x'} & \neg A^x_{x'} & \end{array}}$$

$$\frac{\{x : A\} \not\subset \{x : B\}}{\begin{array}{c} \text{an-element}(f(\vec{x})) \\ A^x_{f(\vec{x})} \\ \neg B^x_{f(\vec{x})} \end{array}}$$

Here x' is a new free variable and $f(\vec{x})$ is a Skolem term where $\vec{x}$ is the vector of free variables occurring in the formula $\{x : A\} \not\subset \{x : B\}$.

Notice that both in the case of equality and subset, the version of the rule that applies to positive formulas introduces a new free variable. The reason for this is that these rules abbreviate a sequence of rules, one of which is the γ-rule. Because of this, we must allow these rules to be applied more than once to a single formula.

Neither of the above rules for equality or $\subset$ are currently wired into the IPR program. Currently, IPR handles reasoning about $\subset$ by including the definition of $\subset$ in the knowledge base and fetching it automatically using the ε-rule [109].

Restricted Paramodulation It appears that some sort of controlled paramodulation (i.e., equality substitution) rule would be very useful in order to trigger the application of comprehension and extensionality. The IPR program uses a very restricted version of paramodulation called Brown's rule [39, 109] that only eliminates Skolem terms. A specific liberalization of this rule would be useful. In particular, in addition to Brown's rule, paramodulation should be applied any time its application will allow the application of one of the comprehension or extensionality rules described above.

Such a well controlled paramodulation rule is still relatively inexpensive and should be very powerful in practice. This rule has not yet been implemented in IPR but experiments by hand have illustrated its usefulness.

The use of this restricted paramodulation rule would let us prove the examples mentioned previously even if they are written in their more familiar form. For example, if the

definition

$$\{a, b\} = \{y : y = a \vee y = b\}$$

is in the knowledge base, then we can prove

$$\text{an-element}(a) \supset a \in \{a, b\}$$

whereas without this rule, we proved the uglier formula

$$\text{an-element}(a) \supset a \in \{y : y = a \vee y = b\}.$$

The formula, an-element$(a) \supset a \in \{a, b\}$, can be proved in first-order logic if the following axiom is used.

$$x \in \{a, b\} \leftrightarrow \text{an-element}(x) \wedge (x = a \vee x = b)$$

We will discuss the comparison of first-order proofs with proofs in the current calculus more in Section 4.2.6.

Notice that this restricted paramodulation rule is applied only when its application allows the application of the comprehension schema or one of the extensionality axioms. Therefore, in the example proofs in this section, we do not always show this step in detail. We will leave terms in their abbreviated form and the reader should be able to understand that the rule is applied in preparation for the application of extensionality or comprehension.

4.2.5 Control Strategies

Now that we have a set of inference rules, we should discuss what strategies should be used in applying them. We will follow the ordinary strategy for applying the α-, δ-, β- and γ-rules. However, any time one of the restricted paramodulation rules or a non-splitting comprehension rules applies, we always apply it. After all of these rules have been applied as much as possible, we apply the splitting comprehension and extensionality rules.

Since γ-rules and the extensionality rules that introduce variables can be and sometimes must be applied more than once to a single formula, for more complex proofs we will need some strategies for deciding when to do this. We will handle these decisions in some simple and

fair manner. For example, after all other rules are applied, then apply a rule that introduces a free variable to the formula to which such a rule has been applied the fewest number of times. When such a decision has to be made, then we say that the proof involves search.

In tableaux and related methods, the problem of deciding which lemmas to generate is well known. Most of the examples we discuss here are provable without any lemma generation.[4] However, in some examples we generate lemmas in order to find shorter proofs. It turns out that if only equality lemmas are generated, then the proof is easy to find in these examples.

4.2.6 Examples

Here we present some examples of theorems whose proofs are found very easily using a systematic application of the rules presented in this dissertation including the rules mentioned in Section 4.2.4, which are not yet implemented in the IPR prover. In each case, we make clear exactly which rules were needed in the proof.

These examples help to illustrate that the comprehension schema described here is very tractable and yet very strong.

Each of these examples can be proved in various ways. We will concentrate here on the proof in the calculus just described. However, we should mention some other techniques for comparison.

Consider a hypothetical example. Suppose that the goal is to prove that

$$(\forall X)(\forall Y)P(X,Y) = P(Y,X)$$

for some function symbol P that is defined as a set as follows

$$(\forall X)(\forall Y)P(X,Y) = \{z : A(z,X,Y)\}$$

where $A(z,X,Y)$ is a formula containing z, X and Y free. In our current approach, we would apply the extensionality rule from Section 4.2.4 followed by the comprehension schema from Section 4.2.1. (We give detailed examples below.)

[4] What we call "lemma generation" is elsewhere called "optimistic folding down" [81].

A second approach is to replace the equality definition with an axiom of the following form.

$$(\forall X)(\forall Y)(\forall z)(z \in P(X,Y) \leftrightarrow \text{an-element}(z) \wedge A(z,X,Y))$$

In order to prove the theorem this way, the axiom of extensionality is also needed. This device brings the theorem down to the level of first-order logic.

If this method is used along with a theorem application rule such as the ε-rule, then the proofs of many of these theorems look very similar to the proofs shown in the examples in this section. However, the proofs are longer and harder to find in first-order logic. The proofs are longer in the first-order case because the rules presented here are just short cuts for the application of the theorems mentioned. The fact that the proofs are slightly longer doesn't seem terrible until you consider what this does to the proof *search*. The fact that the proofs are longer is one of the reasons that they are harder to find. The proofs are harder to find in the first-order case also in part due to the complexity of equality reasoning that would be needed if the restricted paramodulation rules were not used.

Another commonly used approach is to apply the equality definitions and comprehension schema as a preprocessing step rather than on-the-fly when needed as in the current approach [69]. One disadvantage to this approach is that in an interactive prover, the output becomes ugly immediately rather than only becoming ugly when necessary.

Example 4.4 The following theorem has been used as an example for showing the strength of various systems.

$$(a = c \wedge b = d) \leftrightarrow \langle a, b\rangle = \langle c, d\rangle$$

For example, Paulson uses this example to illustrate his ability to write long and somewhat difficult proofs in the Isabella language [96].

This is proved using the following definitions.

$$\begin{aligned} \langle x, y\rangle &= \{\{x\}, \{x, y\}\} \\ \{x\} &= \{y : y = x\} \\ \{x, y\} &= \{w : w = x \vee w = y\} \end{aligned}$$

If this theorem is entered into a prover that implements the rules described in this dissertation, as well as some automatic lemma generation, then the proof is found directly.[5] The user of such a program does not need to be an expert at some unnatural proof language. In fact, the IPR prover prints out an explanation of the proof in English once it is found. In short, rather than requiring the user to learn a new proof language and spend a tremendous amount of time becoming an expert at this language and debugging a proof, a prover using the rules given in this dissertation can explain the proof of this theorem to the user in less than one second.

If the restricted paramodulation rule is used, then the definitions are expanded at just the right time on-the-fly during the proof. For the sake of space, we have omitted the details of the definition expansion steps from the figures. We leave terms in their abbreviated form in Figures 4.4, 4.5 and 4.6.

The theorem can be proved left to right using only Brown's rule. However, in order to prove the direction from right to left, more rules than these are needed.

In the proof of $a = b$, one automatically-generated lemma is also needed for this proof. See Figure 4.4. We indicate automatically generated lemmas by displaying them in brackets as in $[x \neq \{a\}]$.

The first four formulas in Figure 4.4 come from the input by applying α-rules. The first two are needed for the theorem to be provable in this system.

The first split in the tree is the application of the restricted extensionality rule. The next two splits in the tree are applications of the ordinary β-rule.

Formula 3 comes from formulas 1 and 2 by the restricted paramodulation rule. The next splitting is the restricted extensionality applied to formula 3.

Formula 5 comes from formulas 1 and 4 by the restricted paramodulation rule. The next splitting is from the application of the restricted extensionality rule.[6]

[5] The hypotheses an-element(c), an-element(b), an-element(d), an-element($\{a,b\}$), an-element($\{c,d\}$), and an-element($\{a\}$) need to be added before this formula becomes provable in this system. See Figures 4.4, 4.5 and 4.6.

[6] The reader may notice that oppositely signed equalities are considered complementary even if the matching

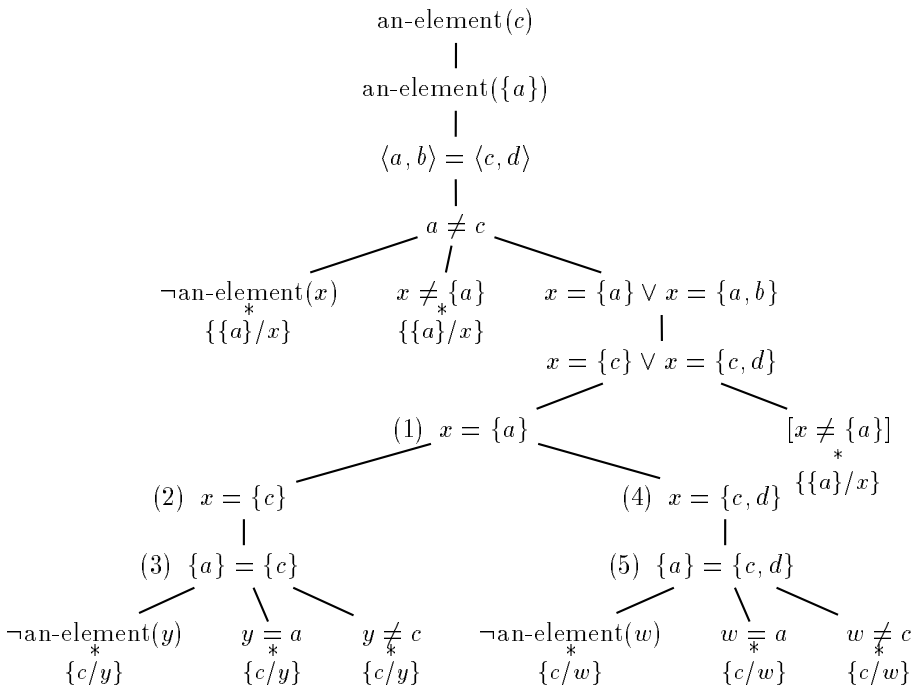

Figure 4.4: The proof of $a = c$ in Example 4.4. The symbols a, b, c and d are Skolem constants.

Before we continue to the explanation of Figures 4.5 and 4.6, we want to show how closely this system relates to the way humans write or explain proofs [110]. Here we describe Figure 4.4 in ordinary language.

> We know that $\langle a, b\rangle = \langle c, d\rangle$ and we want to show that $a = c$. Since $\{a\} \in \langle a, b\rangle$ we know that $\{a\} \in \langle c, d\rangle$ by extensionality. Therefore, $\{a\} = \{c\}$ or $\{a\} = \{c, d\}$ by comprehension.
>
> Suppose $\{a\} = \{c\}$. Since $c \in \{c\}$, we know that $c \in \{a\}$ by extensionality. Therefore, $c = a$ and we are done.
>
> Now suppose $\{a\} = \{c, d\}$. Since $c \in \{c, d\}$, we know that $c \in \{a\}$ by extensionality. Therefore, $c = a$ and we are done.

In Section 5.2, we describe how IPR writes proofs in English. Once the rules from this section are implemented, something like the proof above will be generated automatically from the

terms are on opposite sides of the equalities. This can be handled easily by a unification procedure that recognizes the symmetry of equality or by a ground congruence closure technique. This can also be handled by repeated application of Brown's rule in some cases. None of these methods are expensive.

tableau in a similar way.

The proof of $b = d$ is more complex in this system. See Figure 4.5.

The first formula is the automatically generated lemma from the first part of the proof (i.e., the tableau in Figure 4.4.) The next formulas come from the α-rules applied to the input. The first splitting is the application of the restricted extensionality rule to formula 2. On the middle branch we apply the β-rule and close the leftmost branch using the automatically generated lemma. Then we apply the β-rule again. This time we put the right branch on a separate figure and continue on the left.

Formula 6 comes by the restricted paramodulation rule applied to formulas 4 and 5. Next we apply the restricted extensionality rule followed by a β-rule. Formulas 3 and 12 are complementary. On the left, we apply the restricted extensionality again to formula 6 followed by the β-rule. Formula 9 is formed by Brown's rule applied to formulas 8 and 1. Formula 10 is formed by Brown's rule applied to formulas 7 and 9. Formulas 10 and 3 are complementary.

The first two formulas on Figure 4.6 are repeated from Figure 4.5. They are the automatically generated lemma and the result of the β-rule. Formula 15 comes by the restricted paramodulation rule applied to formulas 14 and 4. Formula 16 comes from the restricted paramodulation rule applied to formulas 13 and 4.

We apply restricted extensionality first to formula 15, then to formula 16. Here y_0 is a Skolem constant. On the left, under formula 18, we apply the β-rule. Formula 24 is the result of Brown's rule applied to formulas 20 and 1. Formulas 18 and 24 are complementary. Formula 25 comes from Brown's rule applied to formulas 18 and 21.

Now, under formula 25, we apply restricted extensionality to formula 2. We apply the β-rule to the two β-formulas. On the left, under formula 29, we apply restricted extensionality. Formulas 31 and 25 are complementary as are formulas 30 and 16.

Under formula 19, we apply the β-rule again. On the left, we use Brown's rule, which makes formulas 26 and 19 complementary. On the right, we apply Brown's rule, which makes formulas 27 and 17 complementary. So the entire tree is closed.

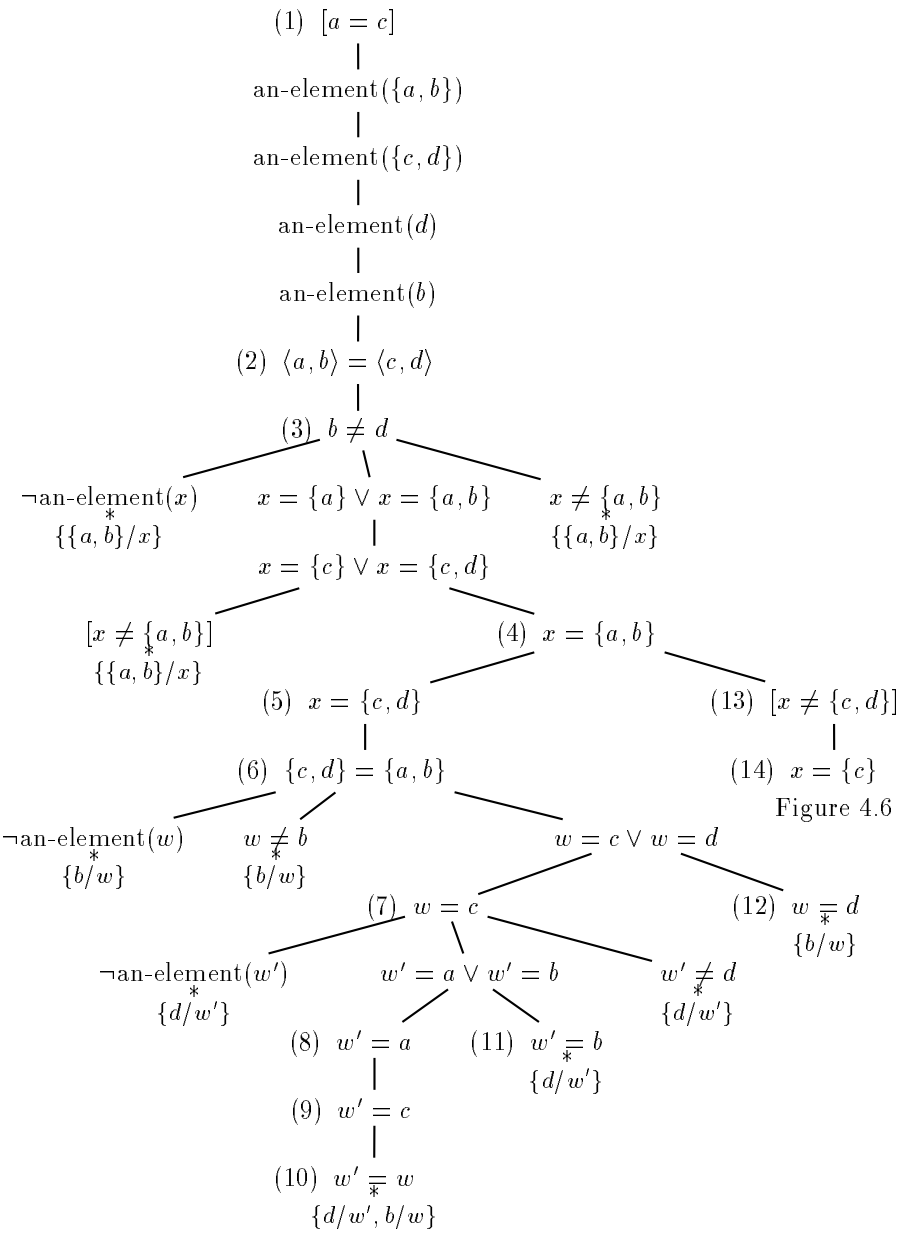

Figure 4.6

Figure 4.5: The proof of $b = d$ in Example 4.4.

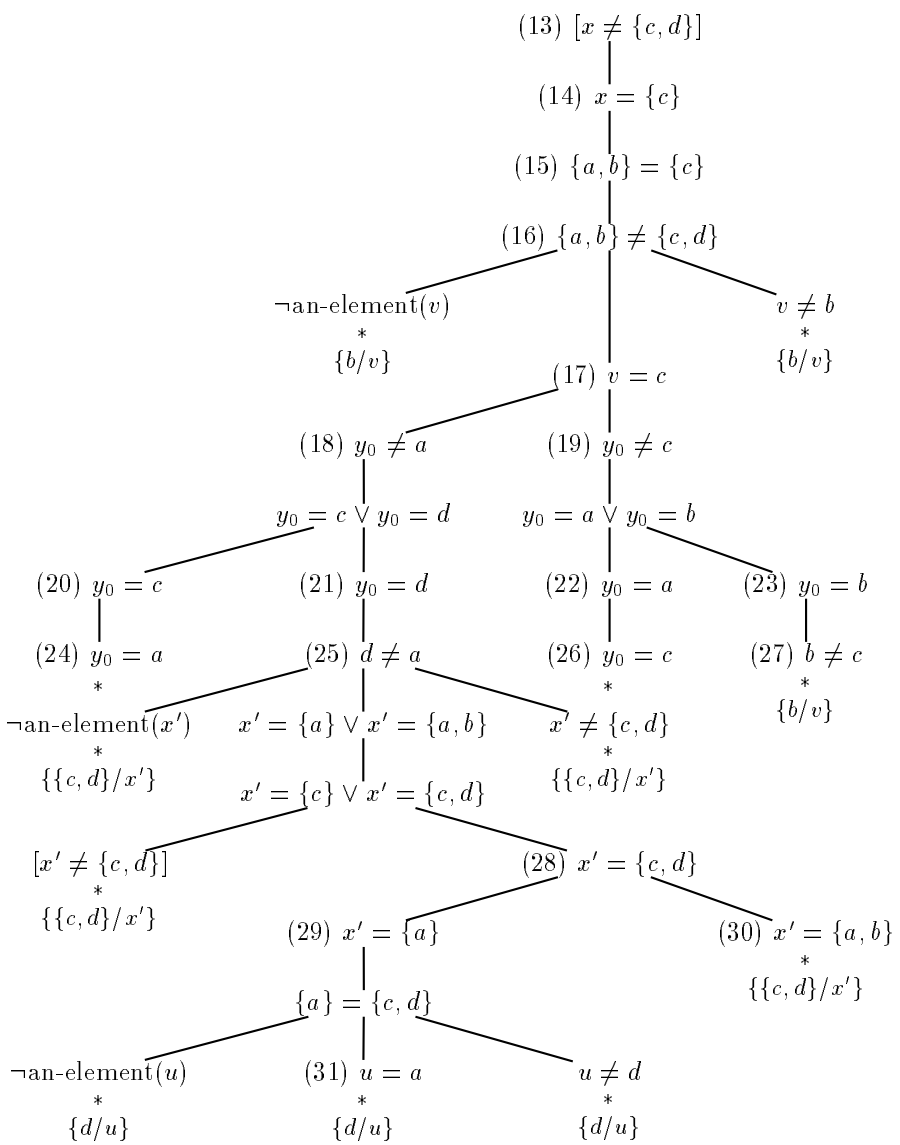

Figure 4.6: Part of the tableau for Example 4.4.

Here again we give a description of the proof in ordinary English.

We know $a = c$ and $\langle a, b\rangle = \langle c, d\rangle$ and we want to show that $b = d$. Since $\{a, b\} \in \langle a, b\rangle$, we know that $\{a, b\} \in \langle c, d\rangle$ by extensionality. Therefore, $\{a, b\} = \{c\}$ or $\{a, b\} = \{c, d\}$ by comprehension.

Suppose $\{a, b\} = \{c, d\}$. Since $b \in \{a, b\}$, we know that $b \in \{c, d\}$ by extensionality. Therefore, $b = c$ or $b = d$ by comprehension.

If $b = d$ then we are done so suppose $b = c$. Since $d \in \{c, d\}$, we know that $d \in \{a, b\}$ by extensionality. Therefore, $d = a$ or $d = b$ by comprehension.

If $b = d$ then we are done so suppose $d = a$. Since $d = a$ and $a = c$, we know that $d = c$. Since $b = c$ and $d = c$, we know that $d = b$ and we are done.

Now suppose $\{a, b\} = \{c\}$ and $\{a, b\} \neq \{c, d\}$ (Figure 4.6.) Since $b \in \{a, b\}$, we know that $b \in \{c\}$ by extensionality. Therefore, $b = c$ by comprehension.

We will use extensionality to show that $\{a, b\} \neq \{c, d\}$, and hence we will have a contradiction.

First suppose $y_0 \in \{c, d\}$ and show that $y_0 \in \{a, b\}$. By comprehension, we know that $y_0 = c$ or $y_0 = d$.

Suppose $y_0 = c$. Since $a = c$ and $y_0 = c$, we know that $y_0 = a$.

Now suppose $y_0 = d$. If we can show that $d = a$ then we will be done.

Since $\{c, d\} \in \langle a, b\rangle$, we know that $\{c, d\} \in \langle a, b\rangle$ by extensionality. Therefore, $\{c, d\} = \{a\}$ or $\{c, d\} = \{a, b\}$ by comprehension. But, $\{c, d\} \neq \{a, b\}$ by hypotheses. Since $\{c, d\} = \{a\}$, then $d = a$ and we are done.

Once the rules from this section are implemented, something like the proof above will be generated automatically from the tableaux in Figures 4.5 and 4.6.

Example 4.5 Now reconsider an example mentioned previously.

$$\mathrm{Pow}(X) \cup \mathrm{Pow}(Y) \subset \mathrm{Pow}(X \cup Y)$$

We mentioned that this is proved without search by the comprehension schema if it is translated down to the language of the classifier without equality as a preprocessing step in Example 4.2. Now we discuss how this abbreviated form is handled in detail.

We will add the following definitions to the knowledge base:

$$\begin{aligned} \mathrm{Pow}(X) &= \{a : (\forall b)(b \in a \supset b \in X)\} \\ X \cup Y &= \{a : a \in X \vee a \in Y\} \end{aligned}$$

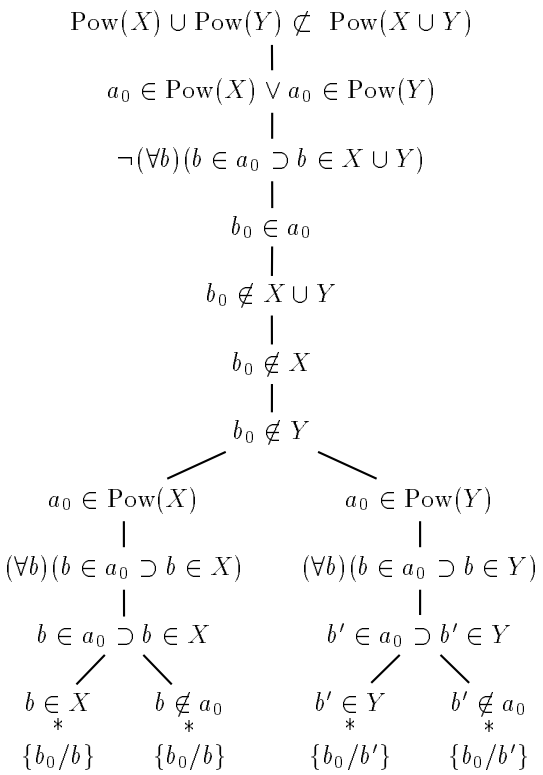

Figure 4.7: The tableau for Example 4.5. The symbols X, Y, a_0 and b_0 are Skolem constants.

The proof in Figure 4.7 shows the proof that is found without search using the rules presented in this dissertation as well as the ordinary analytic tableau rules.[7]

The first formula is the only input. As in the previous example, we do not show all of the definition expansions. Each definition expansion is applied automatically by the very restricted paramodulation rule described in Section 4.2.4.

The next pair of formulas come from the application of the restricted extensionality rule for $\subset$ from Section 4.2.4. A δ-rule and an α-rule are applied in order to create the next pair of formulas. The last pair of formulas on the center branch are created by the application of one of the more efficient comprehension rules. The effect of the application of the more efficient comprehension rule is that the branch containing the formula $\neg$an-element(b_0) is not

[7] The restricted extensionality rule for $\subset$ from Section 4.2.4 is not needed if the definition of $\subset$ is included in the knowledge base. The proof looks almost exactly the same in either case. Proofs using the rule from Section 4.2.4 are slightly shorter.

formed since it would be immediately closed by the application of one of the comprehension branch closure rules.

Now we apply the β-rule to cause the splitting. The proof on each branch goes the same way. We apply comprehension followed by a γ-rule and a β-rule.

The important thing to notice in this example is that there is no search. That is to say, there are no choices and no waste. No formulas are created that are not needed in the proof. There was no wasted time and no choices needed to be made in this example.

Example 4.6 This example is from Paulson's article on the development of set theory in his Isabella prover [96].

$$A \cap B = B \cap A$$

Only the definition of intersection is needed in the proof.

$$A \cap B = \{x : x \in A \wedge x \in B\}$$

The definition of intersection is applied automatically by the restricted paramodulation rule mentioned in Section 4.2.4. After that, the proof is found *without search* by the application of restricted extensionality and the comprehension schema.

See Example 6.5 for a description of a proof of this theorem that does not use any higher-order logic.

4.2.7 Remarks

We first mention that Frank Brown used a very similar rule in his prover for set theory [39]. His rule was more like the one used in Quine's [103] or Zermelo-Fraenkel's [21] set theory. Brown, however, went all the way and implemented all of the axioms of Quine's set theory as inference rules. In the present work, we have tried to be a bit more general purpose by implementing only the comprehension schema. This allows the user to take a step out of first-order logic, and to decide what other axioms of set theory are to be used. With IPR, any selection of set theoretic axioms can be added to the knowledge base and used automatically by the prover.

The method proposed here is a completely automatic method that is tractable, strong and compatible with English explanation [110]. In particular, we build a strong but safe comprehension schema into the tableau calculus. Using the comprehension schema is a much stronger and more direct way of proving theorems than using Gödel's axioms to deconstruct classes. The rules developed here are consistent with the comprehension schema of Kelley [77] or Bernays [21]. The method is compatible with various axiomatizations of set theory and is particularly useful in reasoning in more advanced theories in which the particular axiomatization of set theory is unimportant.

The fact that this rule only goes in one direction makes it unable to solve some difficult problems in set theory. For example, Wos [120] puts forward a challenge for any inference rule claiming to have solved his eighth research problem. This is the proof that $G/\mathrm{Ker}(f) \cong H$ where $f : G \to H$ is an onto homomorphism. The reason this proof is not within the power of the current calculus is that the isomorphism must be *constructed* or *discovered*. The rules presented here only deconstruct classes.

However, Wos' explanation of the challenge might be interpreted in such a way that the current calculus would be strong enough.

> If. . . one were able to suggest a general approach that succeeds in proving various theorems of this type in a reasonable amount of computer time, starting with a fixed database of axioms and a set of lemmas, then we would consider the achievement most notable, whether or not the approach relies on an inference rule for set theory. After all, the rules of the game certainly permit proceeding as mathematicians do: start with some axioms, extend one's knowledge by proving various basic lemmas and by proving various theorems, and then attack diverse questions by selecting what is needed from the accumulated knowledge. Other appropriate test problems include theorems from elementary group theory, point set topology, and number theory.
>
> —Larry Wos [120]

If the usual definitions and lemmas pertaining to this theory were present in the knowledge base of a prover using the ε-rule [109] as well as the rules for set theory presented here and

some slightly more sophisticated equality rules, then the prover could find a proof if the set of lemmas included a mention of the function that is needed in the proof. For example, if the following lemma were included, then the proof could be found.

If $f : G \rightarrow H$ *is an onto homomorphism then there is a function,* $\phi : G/Ker(f) \rightarrow H$ *such that for every* $x \in G$ *and every* $z \in xKer(f)$, $\phi(xKer(f)) = f(z)$.

In this case, the prover would fetch this lemma, which includes a mention of the needed function. Once the needed function is present in the tableau, a prover can prove that it has all of the desired properties. That is to say, the prover can prove that the function is a one-to-one, onto homomorphism. The lemma itself is provable in the current system with more sophisticated equality techniques added.

It does not seem unreasonable to this author to expect to find in a knowledge base theorems such as the one needed to help a prover find this proof. There are, of course, proofs in textbooks that require the prover to construct classes with certain properties when no such class has been mentioned previously in the text. These proofs remain challenging for this approach, other approaches and humans.

Chapter 5

IPR: Control and Interface

> Excellent software is needed to conduct such experimentation adequately. Indeed, the three—theory, software, and applications—are interconnected, and each plays an indispensable role in automated reasoning.
>
> —Larry Wos [119]

> The main goal of research in automated theorem proving is to build programs that are effective in finding or helping to find proofs of theorems from mathematics and other fields of application.
>
> —Bledsoe and Henschen [25]

IPR is the name of a computer program that uses the calculi described in this dissertation. The interface to IPR has some nice features that have been relatively easy to implement due to the fact that the framework for storing and using knowledge is comprehensible. The use of the breadth-first unification strategy in tableau proofs also facilitates human understanding. We describe here the flow of the program when it is working without interaction. Then we describe the interface and what the user can do if he or she desires to be involved during the proof search.

In Section 5.2, we go into some detail about the input, output and interface to the IPR program. We define the language that is used to input formulas to the prover. We describe the possibility for interaction during the theorem-proving process and how to get the output to be in readable English.

Until now, we have discussed an abstract logical calculus for first-order logic. The definition of the calculus does not absolutely determine how the rules will be implemented

and controlled. Below, we address the question of how best to control the search for a proof in this calculus beyond the strategies discussed in Chapter 3. For example, the use of the breadth-first unification strategy in tableau proofs facilitates human understanding and seems to improve the reasoning abilities on some examples.

At the time of this writing, the source code for IPR as well as the input for most of the examples mentioned in this dissertation are available at the following URL.

`ftp://ftp.cs.utexas.edu/pub/bshults/provers/IPR/`

5.1 Control Issues

> Question: Given a specific problem to solve—bearing in mind the tight coupling of representation, inference rule, and strategy—what metarules enable one to simultaneously choose the best representation to use, the most effective inference rule(s) to employ, and the most powerful strategy or strategies to control the chosen inference rule(s)?
>
> —Larry Wos [120]

Until now, we have discussed an abstract logical calculus for first-order logic. The definition of the calculus does not absolutely determine how the rules will be implemented and controlled. In this section, we want to discuss how best to control the search for a proof in this calculus beyond the strategies discussed in Chapter 3.

First we review breadth-first unification. We mention some of its advantages and we describe how to implement Oppacher and Suen's condense strategy with it. Afterward, we discuss strategies for the high-level control of the calculi described in this dissertation.

5.1.1 Breadth-First Unification

IPR uses breadth-first unification for several reasons. In this section, we discuss some of those reasons and explain in some detail how to implement certain strategies with breadth-first unification.

Breadth-first unification allows the proof to be built without ever needing to undo anything that has been done. This is possible due to proof confluence of the tableau-based

procedure. This seems worthwhile since frequently time is the problem and not space. Searching for theorems to apply can take more time than other aspects of reasoning and it seems a shame to waste that time.

Breadth-first unification allows the prover to apply a theorem without applying the substitution σ' across the entire tree until it is discovered that that theorem application is involved in the final proof. In this way, even when IPR does apply the wrong sequent, or the right sequent in the wrong way, it is not prevented from applying the right sequents properly later.

If the user wants to interact with the prover, breadth-first unification provides more flexibility. For example, the interface can present to the user the possible pairs of formulas that may be used to close a particular branch along with the associated substitutions.[1] From this information, the user can use his or her intelligence to choose one of the current possibilities or to allow the prover to expand the branch and find more. This is only possible if the prover does not automatically apply the first branch closure it finds as is done in depth-first strategies.

Another advantage that breadth-first unification provides the interface is that the terms that the user looks at resemble much more closely the terms that the user entered. Since the variables are not instantiated until the entire proof is found, the user never sees terms in places where they will not appear in the final proof.

Breadth-first unification also allows strategies to be applied that cannot be applied if depth-first unification is used. For example, subgoal (or branch) selection [75] becomes a completely different issue. Suppose that for each branch of the tableau, there is a substitution that closes it, however, there is no substitution that closes the entire tableau. In such a case, the prover decides which branch of the tableau to expand further in order to find more closing substitutions. So the branches are sorted dynamically and one is selected. Currently, IPR works on the lightest branch where branches are weighed by the number of closing substitutions but short, simple substitutions are heavier.

[1] This part of the interface has not yet been implemented but these structures all exist when the prover is running. All that is lacking is to give the user easy access to this information.

Implementing strategies with breadth-first unification. As an example, we will show how to implement Oppacher and Suen's condense algorithm [94] using breadth-first unification. With this understanding, it should be easy to see how other strategies would be implemented. There is definitely room for interesting research in the area of applying strategies in the context of breadth-first unification.

First, we show how the basic breadth-first unification can be implemented, then we show what extra bookkeeping is needed to apply the condense strategy.

The following information must be associated with each formula, F:

bindings(F) The substitution on which the existence of F depends. This substitution will have already been applied to F but there may be variables in bindings(F) that did not occur in F before the substitution was applied.

The unification algorithm makes two passes over the tree. During the first pass, all branch closures are found. This pass creates data structures called *initial substitutions*. For simple breadth-first unification, the initial substitution, I, only needs to store the substitution, **subst**(I), that closes the branch and the node, **node**(I), at which I is stored. The initial substitution is stored at the highest node possible, that is at the node of the lowest formula involved in the branch closure.

During the second pass, a combination of one initial substitution from each branch of the tableau is sought such that the substitutions associated with the initial substitutions are compatible (i.e., composable.) During this process, a structure that we call a *unification* is formed. For each unification, U, the following information is needed.

full-subst(U) The substitution associated with U. This is the composition of the substitutions in the initial substitutions used from each branch.

init-subs(U) The list of initial substitutions associated with U.

If a unification is found that combines an initial substitution from each branch, then the entire tableau is closed by that unification.

In order to implement certain strategies, more bookkeeping must be done. Here is one way to implement the condense strategy with breadth-first unification. We will distinguish between two ways that a unification object may take care of a branch of a tableau: a unification may *close* a branch (this has the ordinary meaning) or a unification may *cut* a branch. A branch is cut by a unification object if that branch does not need to be closed by the unification. We will define the notion of cut formally later.

We will sometimes confound a formula and the node on which it resides. We will also need the following definition.

Definition 5.1 (Dominate) We say that a node N *dominates* a node N' if $\langle N, N' \rangle$ is in the transitive closure of the immediate predecessor relation.

The following information needs to be associated with (or computable from) each formula, F.

ancestors-of(F) The formulas on which the existence of this formula depends. For example, any α_1-formula will have the α-formula in this list as well as the ancestors-of(α).

splits(F) The list of nodes in the tableau that dominate F, each of which, N, satisfies the following: (1) N has more than one immediate successor, (2) an element of ancestors-of(F) or F itself occurs on the immediate successor of N that is on the same branch as F. These are the splits in the tableau on which the existence of F depends.

Furthermore, each initial substitution, I, must have the following information accessible or easily computable.

used-formulas(I) The list of formulas involved in this branch closure. For example, if the branch closure is due to a connection on the branch, then the two connected literals are in used-formulas(I); if the branch closure is due to the unification of the terms in a formula of the form $s \neq t$, then only that formula is in used-formulas(I); if the branch is closed due to some equality reasoning, then the formulas in the equality theory

are in used-formulas(I); if the branch is closed by the application of the ε-rule, then ε = used-formulas(I).

init-splits(I) The union of splits(F) for $F \in$ used-formulas(I).

We also need functions to compute or look up the following information about a unification, U.

closed(U) The list of branches closed by U.

need-to-close-up(U) These are the splits of the formulas involved in the unification so far, i.e., the union of init-splits(I) for $I \in$ init-subs(U). Each immediate subnode of the nodes in this list must be labeled with a formula, F such that for some $I \in$ init-subs(U)

$$(\exists F')(I \in \text{init-subs}(U) \wedge F' \in \text{used-formulas}(I) \wedge (F = F' \vee F \in \text{ancestors}(F'))) \quad (5.1)$$

before we say that U closes the entire tableau.

need-to-close-down(U) This is the list of immediate successors of elements of need-to-close-up(U) that do not yet satisfy condition 5.1 for any $I \in$ init-subs(U). The list is ordered left to right.

So every immediate successors N of an element of need-to-close-up(U) is either an element of need-to-close-down(U) or is labeled by a formula satisfying condition 5.1 and not both. We know that U closes the tableau when need-to-close-down(U) is empty and closed(U) is not empty.

Now we can define cut more formally.

Definition 5.2 (cut) A branch B is *cut* by an initial substitution I if B does not pass through the node on which I is stored and the split at which B departs from the branch on which I is stored is not in init-splits(I). A branch B is *cut* by a unification U if it is cut by every initial substitution in init-subs(U).

Now the second pass becomes more complicated. The algorithm is best described recursively. The loop invariants for the main function, Unification-complete(U), include the following.

- There is no element of need-to-close-down(U) above or to the left of any element of closed(U).
- Every branch to the left of the first node in need-to-close-down(U) is either cut or closed by U.
- For each immediate successor, N, of an element of need-to-close-up(U), if $N \notin$ need-to-close-down(U), then the formula at N satisfies condition 5.1 for some $I \in$ init-subs(U).

Pass two This function makes a depth-first left to right survey of all of the initial substitutions in the tableau. For each initial substitution, I, that is found, we first see if all of the branches of the tree to the left of node(I) are cut by I. This is true if and only if none of the immediate successors of the init-splits(I) are to the left of node(I). In this case, we can construct a new unification, U, based on I as follows.

full-subst(U) = subst(I)
init-subs(U) = $\{I\}$
closed(U) = $\{$node(I)$\}$
need-to-close-up(U) = init-splits(I)
need-to-close-down(U) = $\{N : (\exists N')(N' \in$ need-to-close-up(U) and N is an immediate successor of N' and no branch containing node(I) passes through $N\}$
need-to-close-down(U) is sorted left to right.

Once this unification, U, is constructed, we check need-to-close-down(U). If need-to-close-down(U) is empty, then U closes the entire tableau and we are done. Otherwise we call the function Unification-complete(U).

Unification-complete(U) We are given a unification, U, that satisfies the loop invariant. Let N be the first element of need-to-close-down(U). We do a depth-first left to right search for initial substitutions, I, at or under N. For each such I, let σ = Compatible(U, I, N). If $\sigma \neq$fail, then let U' be a copy of U and then set U' = Combine(U', I, σ). If need-to-close-down(U') is now empty, then U' closes the tableau so we return U'. Otherwise, call Unification-complete(U').

Compatible(U, I, N) If the formula, F, at N satisfies condition 5.1 for I and every node to the left of node(I) that is not cut by I is closed by U, then return the composition of full-subst(U) and subst(I) if they are compatible, otherwise return fail.

Combine(U, I, σ)

push$(I,$ init-subs$(U))$

push(node(I), closed$(U))$

full-subst$(U) = \sigma$

merge-into(init-splits(I), need-to-close-up$(U))$

merge-into($\{x : (\exists y)(y \in$ init-splits$(I) \wedge x$ is an immediate successor of $y \wedge$

x is to the right of node$(I))\}$, need-to-close-down$(U))$

sort need-to-close-down(U) left to right.

The IPR program implements this strategy. Once a unification is found that closes the entire tableau, all branches that are cut by the unification are cut from the tableau. By keeping track of the used formulas, IPR is also able to eliminate all formulas that were not directly involved in closing the tableau.

5.1.2 High-level Control

Assuming that IPR has been given a knowledge base and a formula that it is expected to prove with that knowledge, here we describe what IPR does with no interaction from the user

at a very high level. IPR first applies all possible α-rules,[2] then all δ-rules until no more α- or δ-rules apply.[3] If there is a β-rule to be applied, then that is done at this point. After all α-, δ- and β-rules have been applied, IPR applies the breadth-first unification algorithm. If the search for a unification that closes the tableau fails, then IPR selects the branch that it determines needs the most work (according to the discussion at the beginning of Section 5.1.1) and applies a γ-rule to a formula on that branch, if possible. After all α-, δ-, β- and γ-rules (up to a certain limit) have been applied, IPR applies the breadth-first unification algorithm. If this fails to find a closing substitution for the tableau, then IPR selects the branch that it determines needs the most work, ranks all of the possible ways theorems apply to that branch (according to the strategies described in Section 3.3) and applies the theorem that it determines is the most useful using the ε-rule.

After any rule is applied, IPR checks to see if Brown's Rule for equality (Section 4.1) is applicable to one of the new formulas introduced and, if so, applies the rule. Instead of searching the knowledge base when it is time to apply the ε-rule, IPR searches for connections into the knowledge base each time a new literal or γ-formula is introduced.

If the breadth-first unification algorithm finds a proof then the proof discovery procedure stops and the post-processing begins: all formulas and branches that were not involved in the found proof are removed from the tableau and the user is informed that a proof is found.

Example 5.1 Here we proceed through the example of the proof of Theorem 5.19 in Kelley's *General Topology* [77]. Recall that the following three sequents in the knowledge base are sufficient to find the proof.

$$\rightarrow \text{continuous-from-onto}(\pi_a, \prod_A^\tau X, X_a) \tag{5.2}$$

$$\rightarrow \text{open-from-onto}(\pi_a, \prod_A^\tau X, X_a) \tag{5.3}$$

[2] Recall that α-rules simplify a formula and put the simplificants on the same branch. β-rules cause a splitting in the tableau. δ-rules introduce a Skolem term. γ-rules introduce a new variable.

[3] If the theorem involves formulas of the form $a \in \{x : P\}$ or terms such as $\iota_x(P)$, then rules related to higher order logic are applied after all α- and δ-rules are applied. See Section 4.2.

$$\neg(\text{locally-compact}(\textstyle\prod_{A_0}^{\tau} X_0) \supset (\forall a)\text{locally-compact}(X_{0\,a}))$$
$$|$$
$$(1) \quad \text{locally-compact}(\textstyle\prod_{A_0}^{\tau} X_0)$$
$$|$$
$$\neg(\forall a)\text{locally-compact}(X_{0\,a})$$

Figure 5.1: The tableau for Example 5.1 after one step.

$$\left\{ \begin{array}{c} \text{open-from-onto}(f, A, B), \\ \text{continuous-from-onto}(f, A, B), \\ \text{locally-compact}(A) \end{array} \right\} \rightarrow \text{locally-compact}(B) \tag{5.4}$$

The user instructs the prover to prove the following formula:

$$(\forall X)(\forall A)(\text{locally-compact}(\prod_{A}^{\tau} X) \supset (\forall a)\text{locally-compact}(X_a)) \tag{5.5}$$

IPR begins by constructing a tableau rooted at the negation of this formula. It then applies the δ-rule twice converting the universally bound variables in the formula into Skolem constants. Since an α-rule applies to the resulting formula it is applied and the tableau in Figure 5.1 results (with the first few formulas omitted.)

Since formula 1 is a literal, the knowledge base is searched for connections to this formula. (The knowledge base is also searched when a γ-formula is introduced.) Thus, before any more tableau expansion rules are applied, formula 1 is unified with the third formula in the antecedent of sequent 5.4. The substitution needed is $\{\prod_{A_0}^{\tau} X_0/A\}$. Therefore, the information about this connection into the knowledge base is associated with formula 1. We store this information so that we will not have to search the knowledge base using that formula again. If there were other sequents in the knowledge base, then all of them would be searched for formulas that unify with the new formula.

The other new formula in the tableau in Figure 5.1 is a δ-formula and so IPR next applies a δ-rule to produce the tableau in Figure 5.2.

Since the formula introduced by this rule, formula 2, is a literal, the knowledge base is searched for connections to this formula. The only one found is the consequent of sequent 5.4. The substitution needed is $\{X_{0\,a_0}/B\}$. Therefore, this information is associated with formula 2.

$$\neg(\text{locally-compact}(\textstyle\prod^{\tau}_{A_0} X_0) \supset (\forall a)\text{locally-compact}(X_{0\,a}))$$
$$|$$
$$(1) \quad \text{locally-compact}(\textstyle\prod^{\tau}_{A_0} X_0)$$
$$|$$
$$\neg(\forall a)\text{locally-compact}(X_{0\,a})$$
$$|$$
$$(2) \quad \text{locally-compact}(X_{0\,a_0})$$

Figure 5.2: The tableau for Example 5.1 after two steps.

$$\neg(\text{locally-compact}(\textstyle\prod^{\tau}_{A_0} X_0) \supset (\forall a)\text{locally-compact}(X_{0\,a}))$$
$$|$$
$$(1) \ \text{locally-compact}(\textstyle\prod^{\tau}_{A_0} X_0)$$
$$|$$
$$\neg(\forall a)\text{locally-compact}(X_{0\,a})$$
$$|$$
$$(2) \ \neg\text{locally-compact}(X_{0\,a_0})$$
$$\diagup \qquad \diagdown$$
$$(3) \ \neg\text{open-from-onto}(f, \textstyle\prod^{\tau}_{A_0} X_0, X_{0\,a_0}) \qquad (4) \ \neg\text{continuous-from-onto}(f, \textstyle\prod^{\tau}_{A_0} X_0, X_{0\,a_0})$$

Figure 5.3: The tableau for Example 5.1.

No more α-, β- or δ-rules apply because the only unused formulas in the tableau are literals. IPR now computes the best application of the ε-rule. At this time, the knowledge base itself is not searched. Instead, the unifications already found are searched and all compatible combinations are formed and compared. It is easy to see that the best combination, according to the strategies described in Section 3.3, is the combination of the two unifications already mentioned. In this application of the ε-rule, $n+m=2$ and U' has two elements. In this sense, this application is maximal among the possible ε-rule applications satisfying the set-of-support restriction. The set ε will contain the only two literals in the tableau (1 and 2.) We will apply the sequent 5.4 with the substitution $\sigma = \{\prod^{\tau}_{A_0} X_0/A, X_{0\,a_0}/B\}$. Applying the ε-rule, we split the tableau into two branches as shown in Figure 5.3.

Since the two new formulas added to the tableau are literals, the knowledge base is

searched using these formulas. Formula 3 unifies with the formula in sequent 5.3 and Formula 4 unifies with the formula in sequent 5.2. The substitution is $\sigma = \{\prod_{A_0}^{\tau} X_0/A, X_{0\,a_0}/B, \pi_{a_0}/f\}$.

Since no logical rules may be applied, a search is made for a sequent to apply. Both of the current possibilities are equally good and, since the application of either of these sequents closes a branch, both are applied. The left branch is closed by applying the ε-rule with ε containing only formula 3. We use the sequent 5.3 and the substitution $\sigma = \{A_0/A, X_0/X, a_0/a, \pi_{a_0}/f\}$. Therefore, the fact that the left branch is closed by the substitution, $\sigma' = \{\pi_{a_0}/f\}$, using the sequent 5.3 is associated with that node of the tableau in the form of an initial substitution (Section 5.1.1.) The right branch is closed by applying the ε-rule with ε containing only formula 4. We use the sequent 5.2 and the substitution $\sigma = \{A_0/A, X_0/X, a_0/a, \pi_{a_0}/f\}$. Therefore, the fact that the right branch is closed by the substitution, $\sigma' = \{\pi_{a_0}/f\}$, using the sequent 5.2 is associated with that node of the tableau in the form of an initial substitution.

Finally, the breadth-first unification procedure (pass two) searches for a compatible combination of substitutions that will close the entire tree. It finds that the substitutions associated with the nodes on the two branches are identical ($\sigma' = \{\pi_{a_0}/f\}$) and therefore the tree is closed.

At this point, the condense algorithm looks to see if any rule was applied that was not needed for this proof. Since every step was involved in the proof, nothing is done and the user is informed that a proof has been found.

This concludes our discussion of the example. Now we continue to discuss more issues of the high-level control of the calculus.

The IPR prover implements some other methods that seem to be unusual among tableau-based provers. The reason that some of these methods are not used elsewhere is that they only seem to be needed or useful in the context of applying knowledge from a knowledge base or in the context of a prover that aims at ease of use.

IPR also applies non-splitting rules as high (in the semantic tableau sense) in the tree

as possible. That is, it actually *splices* new formulas into the tree at the first point possible so that it is as if they were added to the tree before certain branches were formed. This conserves space, avoids duplication of work and can even allow proofs to be found in the context of using knowledge that could not easily be found otherwise. For example, if a theorem application is applied separately on each branch rather than being spliced in, and that application adds a δ-formula to each branch, then a distinct Skolem function will be created on each branch. From there, it will be impossible for those Skolem functions to be unified to the same variable. However, if the theorem is applied as high as possible and the new δ-formula is spliced into the tree at the highest possible spot, then only one Skolem function is created and the proof can be found. This actually occurs in ordinary examples such as in the proof that the diagonal of a product of a Hausdorff space with itself is closed. (See Example 6.1.)[4]

When IPR applies a theorem, rather than applying some depth-first search of the knowledge base (as in Gazer [11]), IPR ranks the various ways all theorems can be applied.[5] This also differs from the usual method in clausal tableaux in which some fair procedure is applied in clause selection.

If the user is interacting, then he or she can force IPR to apply a theorem in a way that IPR does not consider ideal.[6] The combination of the sensible framework for storing, accessing and applying knowledge allows a human to interact easily. The fact that all IPR output is in English, including formulas in the formal language, facilitates the interaction further. (See Section 5.2.)

Unless the user interacts and tells the prover to undo something, it is not until IPR is completely finished with the proof that the unneeded theorem applications and other unneeded work are undone. At that point the proof is condensed, using Oppacher and Suen's technique [94], and output in English as described in Section 5.2.4.

---

[4] Beckert, Hähnle and Schmitt's δ^{++}-rule could also be used to solve the problem of too many Skolem functions in this particular case [20].

[5] This is implemented in a rather efficient way by storing at each formula in the tableau a list of the formulas in the knowledge base with which it has a connection. Therefore, the knowledge base is only searched once for each new formula in the tableau.

[6] This part of the interface has not yet been implemented. Although the needed information is present when the prover is running, there is not yet an easy way for the user to access it.

It is difficult or impossible to detect waste perfectly, even for humans, except in retrospect. In other words, there is no easy way to be certain that the path that we found through the maze is the shortest. However, it is sometimes not difficult to see that a certain part of our work was really not helpful once we see a solution. This is just as true for mathematicians proving theorems as it is for computers. Only in retrospect—after one or more proofs have been found—can the question of the shortest proof be asked. Even in retrospect, it is difficult to guarantee that the condensed proof is the shortest proof possible.

5.2 The User Interface

We begin by surveying the flow of interaction with the prover. After giving the overall idea of how the prover is used, we give some examples and details of each aspect.

Typically, a person who wants to prove a theorem using IPR goes through a process something like the following. First, he or she should make sure that IPR has enough knowledge to prove the theorem.[7] To do this, he or she either finds the axioms of the theory in the knowledge library that comes with IPR or writes them out himself. Then the user commands IPR to load this knowledge into its knowledge base. With this knowledge is included information that helps IPR to format its output in English.

Now the prover has the needed knowledge and the user can give the statement of the theorem to be proved to IPR. The user may command IPR to stop after each step so that he or she can watch and possibly redirect the efforts of the program. The user may let the prover go and only interrupt it if it seems to take too long. After interrupting, the user can navigate the proof tree that IPR has created and read what rules and theorems were applied. The proof tree navigation process enables the user to understand what has been done and what remains to be done.

Since all of the logical rules applied by IPR, including the theorem application rule (the ε-rule), are related to the way humans explain proofs, the program is able to explain its

[7] We discuss in Section 5.2.3 how the user can sometimes easily detect when there is not enough knowledge in the knowledge base.

reasoning to the user easily.

The user may also take a more active role in the process. The user can even tell IPR every step to take. The user can interact with the prover and find out what some of the possibilities are for the next step and select what he or she thinks is best. The user can also select from the various ways of assigning terms to variables.[8]

If a proof is finally found, the proof tree is condensed [94] to remove all formulas and branches that were not involved in the proof and IPR instantiates the variables in the closing substitution. At this point, the user can continue to navigate the condensed tree. Since the condensed tree only has the information that is needed, it is easy to follow the proof by navigating this tree.

The user can also command IPR to produce an English proof that explains the steps taken and the theorems applied. This English explanation of the proof flows like a proof that a mathematician might write, only with much more detail. This proof explains when a known theorem was applied, which theorem was applied and how it was applied.

In what follows, you will notice that, in its output, IPR maintains certain conventions that allow the user to distinguish between bound variables, free variables and Skolem constants. The conventions assure that the names given to objects introduced by IPR will be distinctly named from the objects introduced by the user. Bound variables and Skolem function symbols will be preceded by the underscore character (_), although free variables are not preceded by the underscore character. In order to further distinguish bound variables from Skolem constants (both of which are preceded by the underscore character), Skolem constants are surrounded by parentheses. Free variables are simply symbols not preceded by the underscore character. A free variable or Skolem function is given a name similar to the name of the bound variable from which it was produced.

These conventions are followed both in the interface and in the English proof output. The author believes that some convention is necessary to avoid confusion between symbols

[8] Not all of this code has been kept up to date since the author has concentrated on using the system as an automatic prover. Therefore, some of these features do not work properly in the latest implementation.

with similar names. With time, the interface should be written to produce LaTeX [80] output.

Ingo Dahn [48] has demonstrated the use of ILF technology to translate the output of IPR into LaTeX source and thence into typeset English. In many of the examples shown in this dissertation, we have applied such a transformation to the IPR output. However, in this section, we show the exact output without such a translation.

5.2.1 The Input Language

Since IPR is not yet able to translate from English into a formal language, the user must enter formulas to be proved in the formal syntax described below.[9] Once these formulas are input, IPR is able to translate everything into English.

Formulas in the input language need to be in fully parenthesized prefix notation. This notation is sometimes called "s-expression syntax" [88]. Here we describe the syntax of the input language including the handling of the classifier, $\{x : A\}$, and the definite descriptor, $\iota_x(A)$.

Definition of term and predicate in the input language. The words "term" and "predicate" are defined recursively in terms of each other.

Definition 5.3 (symbol) A string of characters is a *symbol* provided that every character is either alphabetic, numeric or in the list: $\langle$`+`, `-`, `*`, `/`, `@`, `$`, `^`, `&`, `_`, `=`, `<`, `>`$\rangle$ and provided the first symbol in the string is not a number or `_`.

A symbol is a term in the input language. If $t_1, t_2, \ldots, t_n$ are terms in the input language and f is a function symbol of n arguments in the input language, then (`f` t_1 t_2 $\cdots$ t_n) is a term in the input language. For example, `(the-ordered-pair x (f x))` is a term in the input language where `x` is a variable and `the-ordered-pair` and `f` are function symbols of two and one arguments, respectively.

If $t_1, t_2, \ldots, t_n$ are terms in the input language and P is a predicate symbol of n arguments in the input language, then (`P` t_1 t_2 $\cdots$ t_n) is a formula in the input language. For

[9] Other researchers have worked on the problem of translating English into a formal language for use in automated theorem proving [112].

example, `(a-member-of (the-ordered-pair x (f x)) S)` is a formula in the input language where `x` and `S` are variable symbols and `the-ordered-pair` and `f` are function symbols.

Certain symbols are reserved as predicate symbols: `a-member-of` and `=` are predicate symbols of two arguments and `an-element` and `a-class` are predicate symbols of one argument. The predicate symbol `a-member-of` is the membership predicate symbol. The predicate symbol `=` is the identity predicate symbol. The predicate symbol `an-element` distinguishes objects that can be elements, such as sets or ur-elements, from proper classes. The predicate symbol `a-class` distinguishes classes (including sets) from ur-elements.

If P is a formula in the input language and x is a variable symbol in the input language, then `(the-class-of-all` (x) P`)` and `(the` (x) P`)` are terms in the input language. The term constructor `the-class-of-all` is the classifier. The term constructor `the` is the ι-operator. IPR has mechanisms for handling these predefined operators. See Section 4.2.

If $p_1, p_2, \ldots, p_n$ are formulas in the input language and $x_1, x_2, \ldots, x_m$ are variable symbols in the input language, then `(and` p_1 p_2 $\cdots$ p_n`)`, `(or` p_1 p_2 $\cdots$ p_n`)`, `(not` p_1`)`, `(implies` p_1 p_2`)`, `(if` p_1 p_2 p_3`)`, `(iff` p_1 p_2`)`, `(forall ((`x_1`) (`x_2`) `$\cdots$` (`x_m`))` p_1`)` and `(for-some ((`x_1`) (`x_2`) `$\cdots$` (`x_m`))` p_1`)` are formulas in the input language. `(if` p_1 p_2 p_3`)` stands for "if p_1 then p_2 otherwise p_3."

5.2.2 Input

In order for IPR to format formulas and present proofs in English, it must be given instructions on how non-logical terms and formulas should be formatted. This information can be provided by the user who creates a new theory. The theories that are included with the IPR package already have this information with them. The user is able to change the existing theories to suit his or her tastes.

Description strings. Any formula—be it a theorem, axiom or definition—that is to be loaded into the knowledge base should be accompanied by a *description string*. This is the string IPR will use in the English output of the proof to tell the reader which theorem from the knowledge base was used.

Example 5.2 Suppose we want the definition of a continuous function to be in the knowledge base. The description string might be "the definition of a continuous function." When IPR uses one of the sequents in the knowledge base that was formed from this definition, it refers to the definition using this string. An excerpt of a proof in which IPR uses this definition might look like this:[10]

> Since we know that f is a continuous function from A to B and we are trying to show that f is a function from the class associated with A to the class associated with B we can apply the definition of a continuous function which finishes this branch of the proof.

So the reader can tell which fact in the knowledge base was used at this point in the proof because IPR uses the description string that the user recognizes.

Defining predicates and terms. When a term or predicate is defined, IPR needs to know how to format occurrences of the term or predicate in English. Therefore, the user should accompany definitional formulas with a second string called the *format string*. This string tells IPR how to format any occurrence of the predicate or term being defined.

Example 5.3 In the example of the definition of a continuous function (Example 2.6), the format string might be `"~a is a continuous function from ~a to ~a"`. The `~a` symbols are placeholders for the format strings of the corresponding arguments of the predicate.

Continuing our example, we might have the definition in Figure 5.4(A) loaded into the knowledge base from a file. The only allowed free variables are those occurring in the predicate being defined. When the above definition is evaluated, three sequents are entered into the knowledge base. IPR displays the three sequents as shown in Figure 5.5 (assuming that the other predicates and terms have already had their format strings entered.)

[10] Notice that A and B are topological spaces and hence f is really a *function* from the class associated with A to the class associated with B. Throughout this dissertation we discuss formulas relating to point-set topology and the reader needs to keep in mind the distinction between the topological space and the class associated therewith. The reader who is not familiar with this branch of mathematics may treat all such terms as undefined and concentrate on the reasoning process.

```
(def-predicate (a-continuous-function-from-to f a b)
  (and (a-function-from-to f (top-to-class a) (top-to-class b))
       (forall ((g))
         (implies (open-in g b)
           (open-in (the-set-inverse-image f g) a))))
  (description "the definition of a continuous function")
  (format "~a is a continuous function from ~a to ~a"))
```

(A)

```
(def-term (the-pair a b)
  (the-class-of-all (x) (or (= x a) (= x b)))
  (description "the definition of pair")
  (format "{~a, ~a}"))
```

(B)

Figure 5.4: The input for Example 5.3.

```
Suppose
F is a continuous function from A to B
Then
F is a function from A to B

Suppose all of the following:
1  F is a continuous function from A to B
2  G is open in B
Then
F^{-1}(G) is open in A

Suppose all of the following:
1  F is a function from A to B
2  forall _G  if _G is open in B then F^{-1}(_G) is open in A
Then
F is a continuous function from A to B
```

Figure 5.5: The knowledge for Example 5.3 as IPR displays it.

The definition of a term looks like Figure 5.4(B). This definition adds an equality to the knowledge base.

If the user wants to load the format strings without putting the definitional formula into the knowledge base, that is also possible. Examples are given in the knowledge base that comes with the program.

Ingo Dahn [48] is able to use his ILF technology to translate the output of IPR as shown above into LaTeX source and thence into typeset English. This sort of technology could be incorporated into the IPR system. In many of the examples shown in this dissertation, we have applied such a transformation to the IPR output.

5.2.3 Interaction

The IPR prover is intended to be usable by people who do not know about computer theorem proving techniques. It has accomplished this in several remarkable ways. One very important aspect of this is that it is possible for the user to follow along while a proof is being constructed and understand what has been done and what remains to be done.

The IPR prover also allows the user to give commands about what proof step should be taken next. This would be a hollow statement if it were very difficult for a user to understand what has been happening in the proof so far. However, since the user can navigate the proof that has been constructed so far and understand what has happened and what still needs to be done, the user really can make use of his or her knowledge and expertise to guide the proof.

Being able to understand an incomplete proof has other advantages. For example, if the user has not put enough information into the knowledge base to prove a theorem, it is possible to detect this. I have experienced this first hand. If IPR is not proving a theorem that I think it should prove, I can look at the proof attempt and get to the point where I think it should be finished. At that point, I have been able to see that there is some trivial bit of knowledge that is needed for the proof but that I neglected to enter into the knowledge base.

All this is accomplished by translating each branch of the tableau into a sequent as in Smullyan [113]. All of the unused positive formulas on a branch are put into the antecedent of

```
Description: a statement near the bottom of page 147 of Kelley
Suppose all of the following:
1  F is an open function from A onto B
2  F is a continuous function from A onto B
3  A is locally compact
Then B is locally compact

Description: Theorem 3.2 of Kelley
the Bth projection function of X over A is an open function from
the product of X over the index set A onto the Bth coordinate space

Description: a statement on the top of page 90 of Kelley
the Bth projection function of X over A is a continuous function
from the product of X over the index set A onto the Bth coordinate space
```

Figure 5.6: The knowledge for Example 5.4 as IPR displays it.

the sequent and all of the unused negative formulas on the branch are put into the consequent of the sequent. Thus the tableau is translated into a tree of sequents. IPR allows the user to navigate this tree of sequents by hitting keys such as "up," "down," "next sibling," etc.

Each sequent is printed to the screen as an English explanation of the current hypotheses and goals.

IPR can show the contents of the knowledge base to the user in English. See Figures 6.1, 6.2 and 6.9 for examples of the format IPR uses to display its knowledge. The example of the continuous function was given in the previous section.

Example 5.4 Here we present a simple example of a theorem proved by IPR to give the flavor of the interface and the output. The theorem is taken from John Kelley's *General Topology* [77] (See Example 3.3 in this dissertation.)

Theorem: *If the product of X over the index set A is locally compact then for every a, X_a is locally compact.*

The three sequents shown in Figure 5.6 are in the knowledge base and are used in the proof. They are displayed as IPR displays them.

If the user asks IPR to stop and display after each step taken in the proof, the following sequents will be displayed in sequence each time the user strikes the space key. If IPR is in

```
[1]
Show that
if the product of (_X) over the index set (_A) is locally compact
then for every _B  the _Bth coordinate space is locally compact

[2] PROMOTE
Suppose that
the product of (_X) over the index set (_A) is locally compact
and show that
for every _B  the _Bth coordinate space is locally compact

[3] CONSIDER
Suppose that
the product of (_X) over the index set (_A) is locally compact
and show that
the (_B)th coordinate space is locally compact

[4] (1) APPLY-THEOREM-SPLIT
Suppose that
the product of (_X) over the index set (_A) is locally compact
and show that one of the following is true:
1.  F is an open function from the product of (_X) over the index set
   (_A) onto the (_B)th coordinate space
2.  the (_B)th coordinate space is locally compact

[4] (2) APPLY-THEOREM-SPLIT
Suppose that
the product of (_X) over the index set (_A) is locally compact
and show that one of the following is true:
1.  F is a continuous function from the product of (_X) over the index
    set (_A) onto the (_B)th coordinate space
2.  the (_B)th coordinate space is locally compact
```

Figure 5.7: The interaction for Example 5.4.

interaction mode, then the space key instructs IPR to take a single step in the proof. After the nth step, IPR would display the nth sequent in Figure 5.7 except that the last two sequents are generated at the same time. At any time the user can navigate up and down the tree to understand what is happening in the proof.

At each sequent, the depth of the sequent in the tree is given in brackets and the branch label—(1) and (2) on the last two sequents—is given in parentheses. In complex trees, these labels take forms such as (1.2.1.1.3). This label gives a history of decisions made at choice points to reach the current branch.

The user can navigate this tree and see all of the current hypotheses and all of the

current goals. Since the name of the rule used to obtain the sequent is written above the sequent, the user can tell what changed and how the new formulas were formed. By doing this, the user can tell what has been done in the proof and what remains to be done.

The last two sequents in this proof were formed by the application of the first sequent in the knowledge base. The last two sequents are closed by the application of the last two sequents in the knowledge base with the substitution `the Bth coordinate function` for the variable `F` [109].

The user who runs the program interactively can decide to apply a certain substitution to the tree, to apply a certain bit of knowledge or logical rule, or simply to watch the decisions made by the prover.

Upon finding a proof, the prover condenses it as much as possible (Section 5.1.1), removing steps that were not needed, and explains in English the steps taken including how and when knowledge was applied. This is possible because the rule used for applying knowledge is related to the way humans explain the application of theorems. Once the irrelevant steps have been removed, the user can also navigate the condensed sequent tree and understand the proof by looking at that.

After the proof is found, the proof tree is transformed into what is shown in Figure 5.8.

Notice that on the last two sequents, those formulas that were no longer needed in the proof from that point on are no longer displayed. This minimalist view of the proof tree is much easier to understand than in the presence of the irrelevant formulas.

After the proof is finished, the user can continue to navigate the condensed tree to understand just what was needed in the proof. Since only the needed information is displayed at this point, it is easy to understand how the proof worked. For more details, the user can ask IPR to explain the proof in English.

By interacting, the user can force IPR to apply any known equality, either in the knowledge base or in the hypotheses, in various ways. One common way to force an equality substitution is in the case that a term is defined in the knowledge base and the user wants the

```
[1]
We have shown
if the product of (_X) over the index set (_A) is locally compact
then for every _B, the _Bth coordinate space is locally compact

[2] PROMOTE
If we suppose
the product of (_X) over the index set (_A) is locally compact
then we have shown
for every _B,  the _Bth coordinate space is locally compact

[3] CONSIDER
If we suppose
the product of (_X) over the index set (_A) is locally compact
then we have shown
the (_B)th coordinate space is locally compact

[4] (1) APPLY-THEOREM-SPLIT
We have shown that
the (_B)th projection function is an open function from the product of
(_X) over the index set (_A) onto the (_B)th coordinate space

[4] (2) APPLY-THEOREM-SPLIT
We have shown that
the (_B)th projection function is a continuous function from the
product of (_X) over the index set (_A) onto the (_B)th coordinate space
```

Figure 5.8: The condensed proof tree for Example 5.4.

```
Since we know that
 the product of (_X) over the index set (_A) is locally compact
 and we are trying to show that
 the (_B)th coordinate space is locally compact
we can apply a statement near the bottom of page 147 of Kelley
Now we only need to show that
 the (_B)th projection function of (_X) over (_A) is an open
 function from the product of (_X) over the index set (_A) onto
 the (_B)th coordinate space
and
 the (_B)th projection function of (_X) over (_A) is a continuous
 function from the product of (_X) over the index set (_A) onto
 the (_B)th coordinate space.

1. Since we are trying to show that
 the (_B)th projection function of (_X) over (_A) is an open
 function from the product of (_X) over the index set (_A) onto
 the (_B)th coordinate space
we can apply Theorem 3.2 of Kelley
  which finishes that branch of the proof.

2. Since we are trying to show that
 the (_B)th projection function of (_X) over (_A) is a continuous
 function from the product of (_X) over the index set (_A) onto
 the (_B)th coordinate space
we can apply a statement on the top of page 90 of Kelley
  which finishes that branch of the proof.
```

Figure 5.9: The automatically generated English proof for Example 5.5.

definition to be expanded at an occurrence of the term. The interface makes this very easy. The user simply tells IPR the formula and the occurrence of the term that is to be replaced in the formula. Other equalities in the current hypotheses can be applied in the same way.

5.2.4 Output

When a proof is found, it is condensed and can be output in English.

Example 5.5 Figure 5.9 shows an excerpt (the end) of the English proof output by IPR after the proof in Example 5.4 completed.

This is not intelligent natural language generation but merely the application of templates. Each logical rule, as well as each of the various manifestations of the ε-rule, has a template associated with it. The particular formulas involved in a particular application of a

rule are put into the template and the proof is written. Over the years, as IPR has developed, the templates have changed in order to improve the look of the proofs it produces. The proof example given above was produced by one of the oldest templates. The examples in Chapter 6 use newer templates.

Chapter 6

Examples

> Real problems must be used as test problems to evaluate the worth of an idea; syntactic problems can be used as illustrations, and can also be used for experiments that may lead to important discoveries.
>
> —Larry Wos [120]

IPR has independently found the proofs of many theorems in various fields of mathematics in the presence of relatively interesting knowledge bases. By independently, we mean that the user only entered the needed formulas and the formula to be proved. The user provided no guidance during the proof. That is to say, IPR has been rather successful in solving the kind of problem we have been discussing in this dissertation. As evidenced by the examples in this section, IPR's methods of storing, selecting and applying knowledge as well as the methods for controlling search have promise.

In this chapter, several examples are given of problems solved by the IPR prover from various areas of mathematics. This will give an idea of the kind of reasoning that IPR is able to do.

There are many theorems that IPR cannot prove independently within a reasonable amount of time and memory. Some of the difficulties that remain are discussed in Section 8.3.

In some of the examples, we give some detail about the input and output of the prover. Please see other sections for examples pertaining to specific rules or interface issues as they are dealt with. For example, Section 4.2.6 contains many examples of proofs found automatically in set theory.

Several of these examples were published previously [108].

Example 6.1 The first example we present was one of the main motivating challenges during the recent development of the prover. The proof itself is quite complex and there is some necessary branching involved.

Here is the statement in English.

If S is Hausdorff, then the diagonal of $S \times S$ is closed.

Although this example has been published and an input file has been distributed to many researchers, no other prover has successfully proved this theorem.[1] Among others, Ingo Dahn has tried to prove this example using several resolution and tableau automated theorem proving systems including Otter and Setheo.[2]

The theorem can be found, for example, as an exercise in Chapter 1 of Abraham-Marsden-Ratiu's text *Manifolds, Tensor Analysis, and Applications* [1]. We want to prove

$$(\forall S)(\text{Hausdorff}(S) \supset \text{closed-in}(\text{top-to-class}(\text{diagonal}(S \times_\tau S)), S \times_\tau S))$$

We present here a simple and natural formalization of the needed knowledge. The following eight textbook-level theorems turn into 12 sequents in the knowledge base. These sequents are all needed in the shortest proof that IPR (or this author) could find in the given theory.[3]

the definition of Hausdorff:

$$(\forall X)(\text{Hausdorff}(X) \supset (\forall A)(\forall B)((A \in \text{top-to-class}(X) \wedge B \in \text{top-to-class}(X) \wedge A \neq B) \supset (\exists G1)(\exists G2)(\text{open-in}(G1, X) \wedge \text{open-in}(G2, X) \wedge A \in G1 \wedge B \in G2 \wedge \text{disjoint}(G1, G2))))$$

the definition of closed:

$$(\forall X)(\forall A)((\forall y)(y \in \text{top-to-class}(X) \wedge y \notin A) \supset (\exists G)(y \in G \wedge \text{open-in}(G, X) \wedge \text{disjoint}(G, A))) \supset \text{closed-in}(A, X)$$

[1] In the published version of this example, there was an error that had to be corrected in an errata. However, the distributed input file was correct.

[2] Ingo Dahn: personal communication.

[3] Notice that $\times_\tau$ is the product topology on the product of topological spaces, whereas $\times$ is the simple cartesian product of sets. Also diagonal($S \times_\tau S$) represents a subspace (rather than a subset) of $S \times_\tau S$.

part of the definition of product of topological spaces:
$$(\forall X)(\forall S)(\forall T)(X \in \text{top-to-class}(S \times_\tau T) \supset$$
$$(\exists A)(\exists B)(A \in \text{top-to-class}(S) \wedge B \in \text{top-to-class}(T) \wedge X = \langle A, B\rangle))$$

the definition of product topology:
$$(\forall X)(\forall Y)(\forall A)(\forall B)((\text{open-in}(A, X) \wedge \text{open-in}(B, Y)) \supset$$
$$\text{open-in}(A \times B, X \times_\tau Y))$$

the definition of diagonal:
$$(\forall X)(\forall S)(X \in \text{top-to-class}(\text{diagonal}(S \times_\tau S)) \leftrightarrow$$
$$(\exists A)(A \in \text{top-to-class}(S) \wedge X = \langle A, A\rangle))$$

the axiom of ordered pairs:
$$(\forall A)(\forall B)(\forall C)(\forall D)(\langle A, B\rangle = \langle C, D\rangle \supset (B = D \wedge A = C))$$

the definition of disjoint:
$$(\forall A)(\forall B)(\text{disjoint}(A, B) \leftrightarrow \neg(\exists Y)(Y \in A \wedge Y \in B))$$

the definition of product:
$$(\forall X)(\forall S)(\forall T)(X \in S \times T \leftrightarrow (\exists A)(\exists B)(A \in S \wedge B \in T \wedge X = \langle A, B\rangle))$$

These formulas turn into the sequents shown in Figures 6.1 and 6.2. They are displayed there as IPR displays them when given the appropriate format strings. (See Section 5.2.2.)

To handle the equality reasoning in this proof, only Brown's rule and the $\varepsilon_=$-rule are required (Section 4.1.3.) This equality reasoning in the fetcher is not absolutely needed for the proof but its use helps cut down the size of the search space in this case.

The very impressive fact is that IPR is still able to find a proof even when more information is added to the knowledge base. For example, if the high-level $\supset$ symbol is replaced by the symbol $\leftrightarrow$ in the first three formulas (the definitions of Hausdorff, closed and product of topological spaces) then IPR is still able to find the proof. These changes add three sequents to the knowledge base and each of these sequents is very closely related to the proof but none of the information is used in the proof. The addition of this information makes the problem much more difficult.

The condensed version of the shortest proof found by IPR is complex and involves 4 branches and gets to a depth of 29. No cut rule or any simulation thereof is used in the proof. During the search, the tree created by IPR becomes enormous and branches tremendously. This proof requires a little thought for a mathematician and particularly for a student.

1 Description: the definition of Hausdorff
Suppose all of the following:
1. X is a Hausdorff space
2. A is a member of the class associated with X
3. B is a member of the class associated with X
Then one of the following is true:
1. for some $G1$ and $G2$, $G1$ is open in X and $G2$ is open in X and A is a member of $G1$ and B is a member of $G2$ and $G1$ and $G2$ are disjoint
2. $A = B$

2 Description: the definition of closed
Suppose
for every Y, if Y is a member of the class associated with X and it is not the case that Y is a member of A then for some G, Y is a member of G and G is open in X and G and A are disjoint
Then A is a closed subset of X

3 Description: part of the definition of product of topological spaces
Suppose
X is a member of the class associated with the product topological space of S and T
Then
for some A and B, A is a member of the class associated with S and B is a member of the class associated with T and $X = \langle A, B \rangle$

4 Description: the definition of product topology
Suppose all of the following:
1. A is open in X
2. B is open in Y
Then
the cartesian product of A and B is open in the product topological space of X and Y

5 Description: the definition of diagonal
Suppose all of the following:
1. A is a member of the class associated with S
2. $X = \langle A, A \rangle$
Then X is a member of the class associated with the diagonal of the product topological space of S and S

6 Description: the definition of diagonal
Suppose
X is a member of the class associated with the diagonal of the product topological space of S and S
Then
for some A, A is a member of the class associated with S and $X = \langle A, A \rangle$

Figure 6.1: The first part of the knowledge base formed for Example 6.1.

7 Description: the axiom of ordered pairs
Suppose
$\langle A, B\rangle = \langle C, D\rangle$
Then
$B = D$

8 Description: the axiom of ordered pairs
Suppose
$\langle A, B\rangle = \langle C, D\rangle$
Then
$A = C$

9 Description: the definition of disjoint
One of the following is true:
1. A and B are disjoint
2. for some Y, Y is a member of A and Y is a member of B

10 Description: the definition of disjoint
The following are contradictory:
1. A and B are disjoint
2. Y is a member of A
3. Y is a member of B

11 Description: the definition of product
Suppose all of the following:
1. A is a member of S
2. B is a member of T
3. $X = \langle A, B\rangle$
Then
X is a member of the cartesian product of S and T

12 Description: the definition of product
Suppose
X is a member of the cartesian product of S and T
Then
for some A and B, A is a member of S and B is a member of T and $X = \langle A, B\rangle$

Figure 6.2: The second part of the knowledge base formed for Example 6.1.

(1) Hausdorff(S_0)

|

(2) $\neg$closed-in(top-to-class(diagonal($S_0 \times_\tau S_0$)), $S_0 \times_\tau S_0$)

| $\varepsilon = \{(2)\}, s = \boxed{2}$

$\neg(\forall Y)((Y \in \text{top-to-class}(S_0 \times_\tau S_0) \wedge Y \notin \text{top-to-class}(\text{diagonal}(S_0 \times_\tau S_0))) \supset$
$(\exists G)(Y \in G \wedge \text{open-in}(G, S_0 \times_\tau S_0) \wedge \text{disjoint}(G, \text{top-to-class}(\text{diagonal}(S_0 \times_\tau S_0)))))$

|

(3) $Y_0 \in \text{top-to-class}(S_0 \times_\tau S_0)$

|

(4) $Y_0 \notin \text{top-to-class}(\text{diagonal}(S_0 \times_\tau S_0))$

|

$\neg(\exists G)(Y_0 \in G \wedge \text{open-in}(G, S_0 \times_\tau S_0) \wedge \text{disjoint}(G, \text{top-to-class}(\text{diagonal}(S_0 \times_\tau S_0))))$

|

$\neg(Y_0 \in G \wedge \text{open-in}(G, S_0 \times_\tau S_0) \wedge \text{disjoint}(G, \text{top-to-class}(\text{diagonal}(S_0 \times_\tau S_0))))$

| $\varepsilon = \{(3)\}, s = \boxed{3}$

$(\exists A)(\exists B)(A \in \text{top-to-class}(S_0) \wedge B \in \text{top-to-class}(S_0) \wedge Y_0 = \langle A, B \rangle)$

|

(5) $A_0 \in \text{top-to-class}(S_0)$

|

(6) $B_0 \in \text{top-to-class}(S_0)$

|

(7) $Y_0 = \langle A_0, B_0 \rangle$

| (4), (7), Brown

(8) $\langle A_0, B_0 \rangle \notin \text{top-to-class}(\text{diagonal}(S_0 \times_\tau S_0))$

| $\varepsilon = \{(8), (5)\}, s = \boxed{5}$

(9) $\langle A_0, B_0 \rangle \neq \langle A_0, A_0 \rangle$

Figure 6.3: The top of the tableau for Example 6.1.

In Figures 6.3–6.5, we show the condensed tree constructed by IPR after proving this theorem automatically. IPR often applies α-rules as a processing step. Therefore, sometimes α-formulas will not appear in the tableaux.

In the following description, we do not specify the substitutions that are applied for each ε-rule. It will always be clear from the context. We only indicate when there is a nontrivial σ'.

In Figure 6.3, formulas 1 and 2 are constructed by applying the ordinary analytic tableau rules to the input formula. The next formula is added to the tableau by the application of the definition of a closed set. The next three formulas are formed by the application of the δ-rule (creating the constant Y_0) and the α-rule. Next, the variable G is created by the

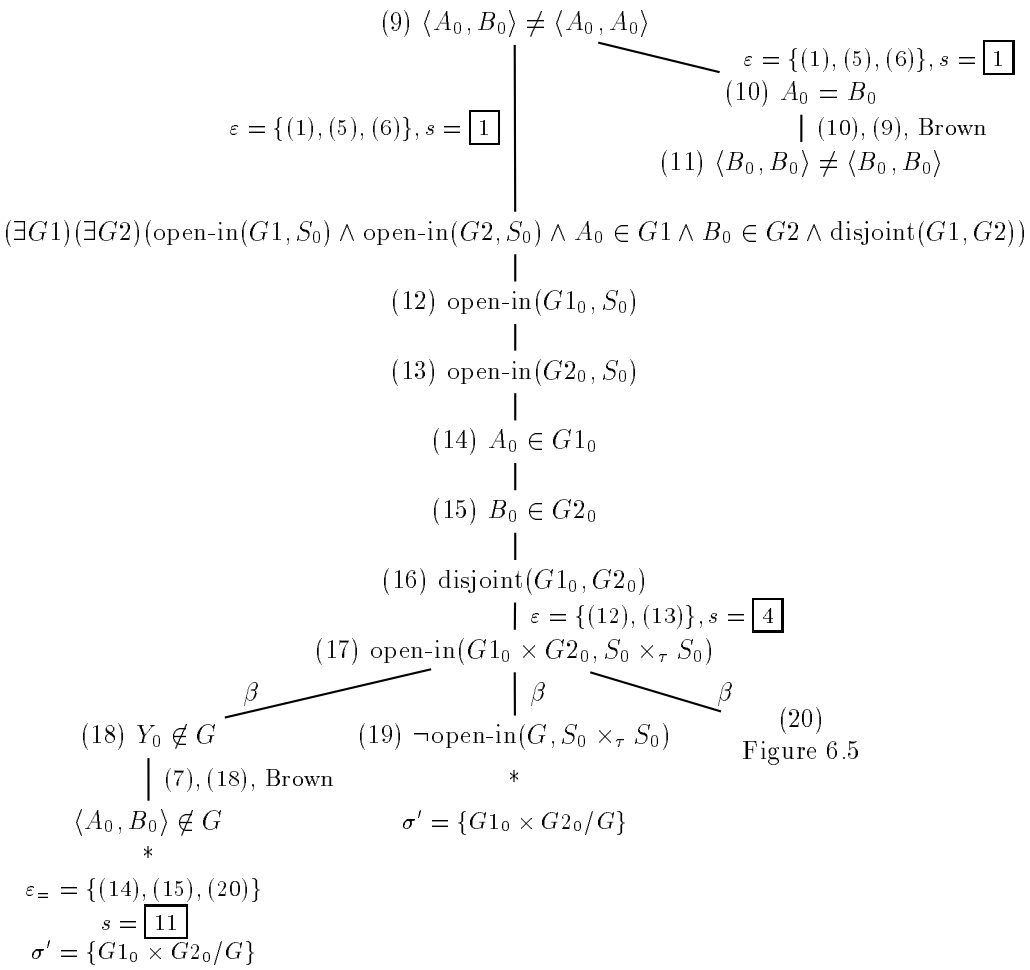

Figure 6.5

Figure 6.4: The middle of the tableau for Example 6.1.

application of the γ-rule. We will wait to apply the β-rule to this formula.[4]

Then we apply part of the definition of product of topological spaces followed by the creation of formulas 5, 6 and 7 by the application of the δ-rule (creating the constants A_0 and B_0) and the α-rule. Now we make an equality substitution and then apply the definition of diagonal.

This tableau is continued in Figure 6.4.

[4] In fact, IPR applies the β-rule right away but then splices as many new formulas as possible above the split. This gives IPR more flexibility, keeps the space requirements down and makes the output look better.

Figure 6.4 begins with the application of the definition of Hausdorff. The right-hand branch is immediately closed by applying Brown's rule and then unifying the terms in the equality goal with the empty substitution.

On the left-hand branch, we apply the δ-rule (creating the new constants $G1_0$ and $G2_0$) along with the α-rule to introduce formulas 12–16. Formula 17 is created by the application of the definition of product topology.

Now we finally apply the β-rule to the β-formula from Figure 6.3. To the leftmost branch we apply an equality substitution. That branch is closed by the application of the definition of product. In this case, $\sigma' = \{G1_0 \times G2_0/G\}$. (So far, in every application of the ε-rule, $\sigma' = \emptyset$.) This is a case in which the $\varepsilon_=$-rule is applied (Section 4.1.3.) In particular, rather than adding a branch containing the formula $\langle A_0, B_0\rangle \neq \langle A_0, B_0\rangle$ to the tableau, the fetcher successfully unifies the terms in the equality and composes the unifier ($\emptyset$) with the substitution found so far in the ε-rule application. The middle branch is closed by the same substitution, $\sigma' = \{G1_0 \times G2_0/G\}$, by unifying formulas 17 and 19.

The tableau for the rightmost branch of that split is shown in Figure 6.5.

The remaining part of the proof is certainly not optimal. This just happens to be the first way of proving the theorem that IPR detected.

In Figure 6.5, we first apply the definition of disjoint and then the δ- and α-rules to the resulting formula and create the new constant Y_1. Next we apply the definition of diagonal and again the δ- and α-rules to obtain formula 23, which contains the new constant A_1. (Notice here that only α_2 is needed in the proof.)

The mathematician might write, "Since we are trying to show that G and $S_0 \times S_0$ are disjoint, suppose $y_0 \in G$ and $y_0 \in S_0 \times S_0$ and look for a contradiction. Since $y_0 \in S_0 \times S_0$, $y_0 = \langle a_1, a_1\rangle$ for some $a_1 \in S_0$."

Next, we apply Brown's rule and then the definition of product with $\sigma' = \{S \times T/G\}$. Here S and T are new variables in the tableau introduced by the ε-rule.

$$
\begin{array}{c}
(20)\ \neg\text{disjoint}(G, \text{top-to-class}(\text{diagonal}(S_0 \times_\tau S_0))) \\
\mid \varepsilon = \{(20)\}, s = \boxed{9} \\
(\exists Y)(Y \in G \wedge Y \in \text{top-to-class}(\text{diagonal}(S_0 \times_\tau S_0))) \\
\mid \\
(21)\ Y_1 \in G \\
\mid \\
(22)\ Y_1 \in \text{top-to-class}(\text{diagonal}(S_0 \times_\tau S_0)) \\
\mid \varepsilon = \{(22)\}, s = \boxed{6} \\
(\exists A)(A \in \text{top-to-class}(S_0) \wedge Y_1 = \langle A, A\rangle) \\
\mid \\
(23)\ Y_1 = \langle A_1, A_1\rangle \\
\mid (21), (23), \text{Brown} \\
(24)\ \langle A_1, A_1\rangle \in G \\
\mid \varepsilon = \{(24)\}, s = \boxed{12}, \sigma' = \{S \times T/G\} \\
(\exists A)(\exists B)(A \in S \wedge B \in T \wedge \langle A_1, A_1\rangle = \langle A, B\rangle) \\
\mid \\
(25)\ A_{S,T} \in S \\
\mid \\
(26)\ B_{S,T} \in T \\
\mid \\
(27)\ \langle A_1, A_1\rangle = \langle A_{S,T}, B_{S,T}\rangle \\
\mid \varepsilon = \{(27)\}, s = \boxed{7} \\
(28)\ A_1 = B_{S,T} \\
\mid (27), (28), \text{Brown} \\
(29)\ \langle B_{S,T}, B_{S,T}\rangle = \langle A_{S,T}, B_{S,T}\rangle \\
\mid \varepsilon = \{(29)\}, s = \boxed{8} \\
(30)\ B_{S,T} = A_{S,T} \\
\mid (26), (30), \text{Brown} \\
(31)\ A_{S,T} \in T \\
* \\
\varepsilon = \{(16), (25), (31)\} \\
s = \boxed{10} \\
\sigma' = \{G1_0/S, G2_0/T\}
\end{array}
$$

Figure 6.5: The bottom of the tableau for Example 6.1.

The δ- and α-rules produce formulas 25–27 in which $A_{S,T}$ and $B_{S,T}$ are new Skolem terms. The axiom of ordered pairs gives us formula 28. We use this to apply Brown's rule to formula 27 and then apply the axiom of ordered pairs again. Formula 30 is just what we have been waiting for because now we can use it to find a contradiction using the definition of disjoint.

In Figure 6.6 we show how IPR displays the top of Figure 6.5 in its interaction window once it has been condensed. In this display, formulas are displayed only until the last time they are needed. Thus the user knows that the formulas that disappeared in the new sequent are those that were used to form the new formulas in the new sequent. However, sometimes the used formulas will remain in the new sequent because they will be needed again later in the proof. Remember that it is only possible to detect this in retrospect in cases such as γ-, ε- and equality rules. In the case of α-, δ and β-rules, the used formula can always be eliminated. Also notice that the variables are instantiated in the proof tree once the proof is found.

It is not possible to pin down a single reason IPR is able to prove this. Some experts believe that the non-clausal form helps. I think it is a combination of things including the strategies and breadth-first unification. In a proof this complex with this many possible ways to search, throwing away and rebuilding the tableau become expensive. If breadth-first unification is used, then nothing is thrown away. This is not enough, of course. If nothing is thrown away then there will be a terrible space problem unless the prover uses strategies to guide the search.

The proof follows rather closely the proof that a student would find for this theorem. Of course, the student would write it in English along with a picture. The proof that IPR writes out in English is long and rather boring in this case. The problem of making long proofs output by the prover read well is difficult [74, 110]. We give an example below of the English output of IPR.

Example 6.2 Here is an example from the theory of vector spaces. It is Proposition 2.4.3 of Bishop and Goldberg's text, *Tensor Analysis on Manifolds* [24].

$$(\forall W)(\forall V)((\text{a-vector-subspace}(W,V) \wedge \text{a-vector-space}(V)) \supset \\ (\exists E)(\exists F)(\text{basis-of}(E \cup F, V) \wedge \text{basis-of}(E, W)))$$

```
[17] (1.3)
AND-SPLIT
If we suppose:
(_G1) and (_G2) are disjoint
then we have shown:
the cartesian product of (_G1) and (_G2) and
the class associated with the diagonal of
the product topological space of (_S) and (_S) are disjoint

[18] (1.3)
APPLY-THEOREM
The following are contradictory:
1  (_G1) and (_G2) are disjoint
2  for some _Y
    _Y is a member of the cartesian product of (_G1) and (_G2)
    and
     _Y is a member of the class associated with
     the diagonal of the product topological space of (_S) and (_S)

[19] (1.3)
CONSIDER
The following are contradictory:
1  (_G1) and (_G2) are disjoint
2  (_Y0) is a member of the cartesian product of (_G1) and (_G2)
3  (_Y0) is a member of the class associated with
   the diagonal of the product topological space of (_S) and (_S)

[20] (1.3)
APPLY-THEOREM
The following are contradictory:
1  (_G1) and (_G2) are disjoint
2  (_Y0) is a member of the cartesian product of (_G1) and (_G2)
3  for some _A
    _A is a member of the class associated with (_S)
    and (_Y0) = <_A, _A>

[21] (1.3)
CONSIDER
The following are contradictory:
1  (_G1) and (_G2) are disjoint
2  (_Y0) is a member of the cartesian product of (_G1) and (_G2)
3  (_Y0) = <(_A0), (_A0)>
```

Figure 6.6: Part of the interaction for Example 6.1.

IPR finds the proof easily using the knowledge base formed from the following five formulas:

the definition of a basis
$$(\forall b)(\forall V)(\text{basis-of}(b,V) \leftrightarrow (\text{lin-ind-subset}(b,V) \wedge b \subset \text{vec-to-class}(V) \wedge \text{spans}(b,V)))$$

Proposition 2.2.5 in Bishop and Goldberg
$$(\forall s)(\forall V)(\forall t)(\text{lin-ind-subset}(s,V) \wedge \text{basis-of}(t,V) \supset (\exists u)(u \subset t \wedge \text{basis-of}(s \cup u, V)))$$

a remark on page 63 of Bishop and Goldberg
$$(\forall A)(\text{a-vector-space}(A) \supset (\exists b)\text{basis-of}(b,A))$$

the definition of a subspace
$$(\forall A)(\forall B)(\text{a-vector-subspace}(A,B) \supset (\text{a-vector-space}(A) \wedge \text{vec-to-class}(A) \subset \text{vec-to-class}(B)))$$

Proposition 2.4.2 in Bishop and Goldberg
$$(\forall W)(\forall V)(\forall e)((\text{a-vector-subspace}(W,V) \wedge e \subset \text{vec-to-space}(W)) \supset (\text{lin-ind-subset}(e,W) \leftrightarrow \text{lin-ind-subset}(e,V)))$$

In fact, IPR is able to find the proof of this theorem even when other related but unnecessary formulas are in the knowledge base.

For the purpose of illustration, we will give a full presentation of IPR's proof of a similar theorem. We want to prove the following formula that is slightly simpler to prove.

$$(\text{subspace}(W_0,V_0) \wedge \text{basis}(U_0,V_0)) \supset (\forall E)(\text{basis}(E,W_0) \supset (\exists F)(\text{basis}(\text{union}(E,F),V_0) \wedge \text{subset}(F,U_0)))$$

This follows from a knowledge base containing the following sequents.

[1] the definition of a basis
$$\text{basis}(B,V) \rightarrow \text{lin-ind}(B,V)$$

[2] Proposition 2.4.2 in Bishop and Goldberg
$$\left\{ \begin{array}{c} \text{subspace}(W,V), \\ \text{lin-ind}(E,W) \end{array} \right\} \rightarrow \text{lin-ind}(E,V)$$

[3] Proposition 2.2.5 from Bishop and Goldberg
$$\left\{ \begin{array}{c} \text{lin-ind}(S,V), \\ \text{basis}(T,V) \end{array} \right\} \rightarrow (\exists U)(\text{subset}(U,T) \wedge \text{basis}(\text{union}(S,U),V))$$

Figure 6.7 shows the tableau constructed for this example.

In this case, we let the English output of the IPR program explain the proof. Figure 6.8 contains IPR's explanation of the proof.

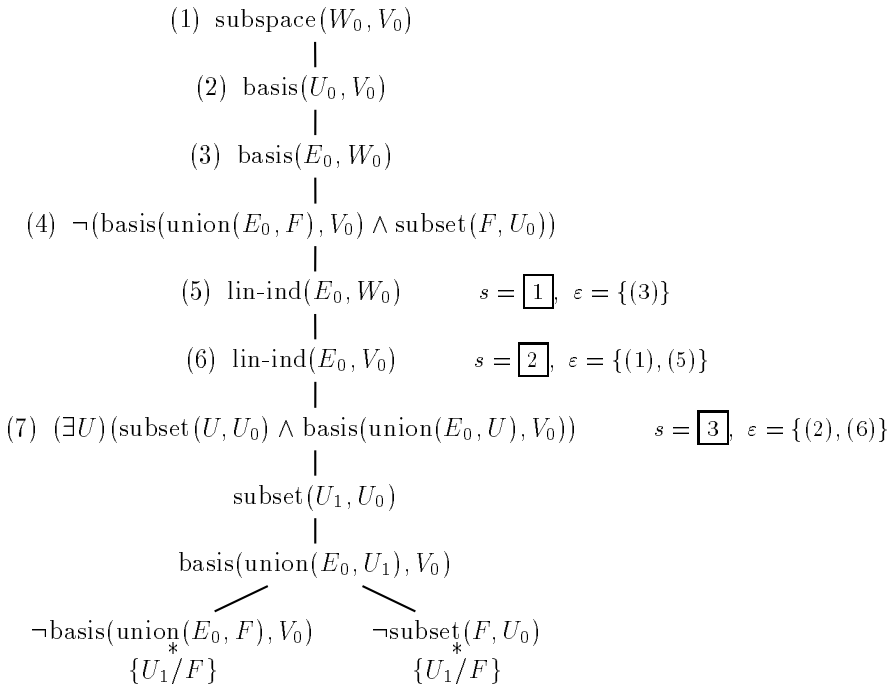

Figure 6.7: The tableau for Example 6.2.

Proof: Since we know that E_0 is a basis of W_0 we can apply the definition of a basis and conclude that E_0 is a linearly independent subset of W_0.
Since we know that W_0 is a subspace of V_0 and E_0 is a linearly independent subset of W_0 we can apply Proposition 2.4.2 in Bishop and Goldberg and conclude that E_0 is a linearly independent subset of V_0.
Since we know that E_0 is a linearly independent subset of V_0 and U_0 is a basis of V_0 we can apply Proposition 2.2.5 from Bishop and Goldberg and conclude that for some U, U is a subset of U_0 and the union of E_0 and U is a basis of V_0.
Let U_1 be such that U_1 is a subset of U_0 and the union of E_0 and U_1 is a basis of V_0.
Split the goal: the union of E_0 and F is a basis of V_0 and F is a subset of U_0 into cases.
1. Since we know that the union of E_0 and U_1 is a basis of V_0 and we are trying to show that the union of E_0 and F is a basis of V_0 we are finished if we substitute U_1 for the variable F.
2. Since we know that U_1 is a subset of U_0 and we are trying to show that F is a subset of U_0 we are finished if we substitute U_1 for the variable F.

Figure 6.8: IPR's proof of Example 6.2.

Example 6.3 The next example is from homotopy theory. It is Corollary 2.2 in Chapter 8 of Munkres' *Topology: A first course* [91]. In this example, we ignore the difference between a topological space and its underlying set since it does not come into play in this proof.

$$(\forall X)(\forall x_0)(\forall x_1)((\text{path-connected}(X) \wedge x_0 \in X \wedge x_1 \in X) \supset \\ \text{isomorphic-groups}(\pi_1(X, x_0), \pi_1(X, x_1)))$$

The knowledge base formed from the following formulas is sufficient although a bit more than necessary for the proof. The proof was found with these theorems and others in the knowledge base by IPR.

The definition of "isomorphic-groups"
$$(\forall A)(\forall B)(\text{isomorphic-groups}(A, B) \leftrightarrow (\exists f)\text{group-isomorphism}(f, A, B))$$

The definition of path connectedness
$$(\forall X)(\text{path-connected}(X) \leftrightarrow (\forall x_0)(\forall x_1)((x_0 \in X \wedge x_1 \in X) \supset \\ (\exists p)\text{path-from-to}(p, x_0, x_1, X)))$$

Theorem 2.1 in Chapter 8 of Munkres
$$(\forall a)(\forall x_0)(\forall x_1)(\forall X)\text{path-from-to}(a, x_0, x_1, X) \supset \\ \text{group-isomorphism}(\widehat{a}, \pi_1(X, x_0), \pi_1(X, x_1))$$

These theorems turn into the five sequents in Figure 6.9. Only three of the five are needed in the proof. All five are displayed in the figure as IPR displays them. IPR is also able to find the proof of this theorem even when other related but unnecessary formulas are in the knowledge base. Here we only show a minimal set of formulas.

In this case we show in Figure 6.10 what the proof tree that the user can navigate after the proof is found looks like. The last sequent in the proof tree is closed by the application of what it means to be isomorphic. At any point, the user can ask for more details about how a sequent was formed (especially when it was a theorem application.) The user may navigate up and down the tree at will.

Example 6.4 Here is an example from the appendix on set theory in Kelley's text on topology [77]. In this example, *all* of the previously-occurring theorems and definitions were present in the knowledge base. This came to 59 sequents including 8 term definitions. The theorem is Theorem 33 in the appendix.

$$(\text{an-element}(x) \wedge z \subset x) \supset \text{an-element}(z)$$

Description: Theorem 2.1 in Chapter 8 of Munkres
Suppose
A is a path from x_0 to x_1 in X
Then
A-hat is an isomorphism from $\pi_1(X, x_0)$ to $\pi_1(X, x_1)$

Description: the definition of path connectedness
Suppose
for all x_0 and x_1, if x_0 is a member of X and x_1 is a member of X then for some P, P is a path from x_0 to x_1 in X.
Then
X is path connected

Description: the definition of path connectedness
Suppose all of the following:
1. X is path connected
2. x_0 is a member of X
3. x_1 is a member of X
Then
for some P, P is a path from x_0 to x_1 in X

Description: what it means to be isomorphic
Suppose
F is an isomorphism from A to B
Then
A and B are isomorphic

Description: what it means to be isomorphic
Suppose
A and B are isomorphic
Then
for some F, F is an isomorphism from A to B

Figure 6.9: The knowledge base sufficient for Example 6.3.

[1]
We have shown:
if X_2 is path connected and x_0 is a member of X_2 and x_1 is a member of X_2 then $\pi_1(X_2, x_0)$ and $\pi_1(X_2, x_1)$ are isomorphic

[2]
PROMOTE
If we suppose all of the following:
1. X_2 is path connected
2. x_1 is a member of X_2
3. x_0 is a member of X_2
then we have shown:
$\pi_1(X_2, x_0)$ and $\pi_1(X_2, x_1)$ are isomorphic

[3]
APPLY-THEOREM
If we suppose:
for some P, P is a path from x_0 to x_1 in X_2
then we have shown:
$\pi_1(X_2, x_0)$ and $\pi_1(X_2, x_1)$ are isomorphic

[4]
CONSIDER
If we suppose:
P_0 is a path from x_0 to x_1 in X_2
then we have shown:
$\pi_1(X_2, x_0)$ and $\pi_1(X_2, x_1)$ are isomorphic

[5]
APPLY-THEOREM
If we suppose:
P_0-hat is an isomorphism from $\pi_1(X_2, x_0)$ to $\pi_1(X_2, x_1)$
then we have shown:
$\pi_1(X_2, x_0)$ and $\pi_1(X_2, x_1)$ are isomorphic

Figure 6.10: The proof tree for Example 6.3.

The proof was found by IPR under these circumstances in a fraction of a second.

Since Kelley does not allow ur-elements in his theory, he uses the term "a set" to refer to what we mean here by "an element." In fact, his definition of a set is "x is a set if and only if for some y, $x \in y$," which is a wired-in axiom of IPR's higher-order reasoning mechanism except IPR uses "an element" in place of "a set."

Here is the main theorem that is needed in the proof. It is Axiom III, Axiom of Subsets, in the appendix.

$$\text{an-element}(x) \supset (\exists y)(\text{an-element}(y) \wedge (\forall z)(z \subset x \ \supset z \in y))$$

One of the comprehension rules is also used to find this proof.

Example 6.5 This example is from Paulson's article on the development of set theory in his Isabella prover [96]. The following theorem is proved completely automatically by IPR.

$$A \cap B = B \cap A$$

Or in the input format:

```
(= (the-intersection-of A B) (the-intersection-of B A))
```

The formulas in Figure 6.11 are used to form the knowledge base. They are displayed there in the form needed as input to IPR. (See Section 5.2.2.)

The definitions of subset and union are put into the knowledge base just to distract the prover from the correct path. IPR finds the proof completely automatically even in the presence of a good bit of unneeded knowledge.

Previously (Example 4.6) we mentioned that this theorem could be proved using the comprehension schema and the other rules presented in Section 4.2. In that case, the definitions of terms such as $A \cap B$ were given as equalities by saying that the intersection of two sets is a certain class. In the current proof, we instead use a theorem that describes what it means to be an element of the intersection of two sets. This device brings the theorem into the realm of first-order logic. Therefore, in this case, no higher-order rules from Section 4.2 are needed.

```
(def-axiom extensionality
  (iff (= x y)
       (forall ((z))
         (iff (a-member-of z x)
              (a-member-of z y)))))

(def-term (a-subset-of a b)
  (forall ((x))
    (implies (a-member-of x a)
      (a-member-of x b))))

(def-theorem subset
  (iff (forall ((x))
         (implies (a-member-of x a)
           (a-member-of x b)))
       (a-subset-of a b)))

(def-theorem union
  (iff (for-some ((b)) (and (a-member-of b c) (a-member-of a b)))
       (a-member-of a (the-union-over c))))

(def-theorem intersection
  (iff (and (a-member-of c a)
            (a-member-of c b))
       (a-member-of c (the-intersection-of a b))))
```

Figure 6.11: The input for Example 6.5.

Chapter 7

Related Work

As indicated in the introduction, a tremendous amount of work has been done already in the field of automated reasoning. It would be impossible to discuss it all in this dissertation. The motivations for the present work—automated reasoning in mathematics together with human understanding—distinguish it from a large amount of the work that has been done by others.

There are some recent projects that share very nearly the motivations of this project. The OMEGA project [74], overseen by Jörg Siekmann, incorporates many components that come together as an excellent automatic proof discovery and explanation system. Parts of OMEGA include PROVERB [73], a system that translates resolution proofs into "assertion-level" proofs and explains them in natural language; LEO/HOTEL [79], which incorporates some set theoretic and equality reasoning power; as well as other components.

Bruno Buchberger's THEOREMA project intends to develop a system that incorporates various special purpose provers each in its respective area. The combination of these reasoning systems will be unified in a single interface for examining proofs in English [40].

Ingo Dahn's ILF project also has similar aims [48]. His system attempts to prove a sentence using a large set of powerful theorem provers and then, upon success, explains the proofs in English.

Many other systems share one of the two motivations of the IPR project. Systems that emphasize interface typically do not include a strong independent inferencing system. On the other hand, most systems that emphasize strong automatic reasoning do not strive to be usable by a non-expert.

We will discuss work related to both of the motivations: automated mathematical reasoning and human interface.

7.1 Reasoning

We already mentioned in Section 2.1 other work related to the choice of language to be used. While higher-order languages are more convenient for expressing mathematical facts, the reasoning in higher-order languages is much less tractable.

Many theorem proving programs use tableaux and related methods. PROTEIN [14], ${}_3T^AP$ [19], lean-T^AP [16], SETHEO [82], MGTP and it variants [66, 65], METEOR [6], Parthenon [35] and PTTP [115] are some of the better known systems.

Most existing systems and calculi use clausal form for storing the theory. Some save some space by breaking a formula only to negation normal form [63]. This form still has the disadvantages associated with Skolem constants.

The non-clausal format in which knowledge is stored as described in this dissertation appears to be a novelty. The Gazer system stores knowledge in a similar but less general form. Gazer essentially assumes that the input is nearly Horn [11].

While clausal form is compatible with the ε-rule—indeed clausal form allows the ε-rule to be applied with the set-of-support restriction (weak connection condition) without losing completeness—we have stressed the advantages of the non-clausal form in Section 2.3.

In some systems, knowledge is stored with additional information supplied by the user, which instructs the prover *how* or *when* to use the knowledge [38, 68]. In this calculus, only the formulas themselves are entered. No indexing or additional information is provided.

The ε-rule is closely related to various strategies used in clausal tableaux [81]. The ε-rule is distinct from existing rules for clausal tableau expansion in various ways. METEOR [6], Parthenon [35], PTTP [115], SETHEO [82] and PROTEIN [14] use calculi closely related to model elimination. Therefore, they obey the connection condition. In a clausal context, the ε-rule only uses the weak connection condition (set-of-support) as in Hähnle's ordered

tableaux [64].

The system ${}_3T^AP$ uses the weak connection condition but considers only one connection at a time. The clausal (and non-clausal) ε-rule takes into account any number of connections from a single clause (or, more generally, a single sequent in the knowledge base) into a given branch. A second apparent distinction between the ε-rule and the A-ordered tableau technique [64] used in ${}_3T^AP$ is that the ε-rule does not add the connected literals to the tableau. This makes little difference if a depth-first search is made of a unification, however, in a breadth-first search, the omission of the extra literals cuts the search space tremendously.

Baumgartner's hyper-tableaux resemble S-tableaux formed by the ε-rule [15]. Indeed, in some proofs the tableaux constructed by the two methods are identical. The main differences are (1) that the ε-rule allows both positive and negative literals to be connected during the rule application, (2) the ε-rule requires only the weak connection condition and (3) hyper-tableaux have range-restrictedness, which allows variables to be universal rather than rigid (or mixed) as in the case of the ε-rule.

A number of good ideas have been developed for reasoning in the presence of a lot of knowledge. Most of the closely related work in automating the use of a knowledge base relies on the use of rewriting or Horn clauses. That is, the existing methods that work well on large knowledge bases do not work well if the knowledge is non-Horn.

Among the most closely-related work is that of Dave Barker-Plummer on his Gazing algorithm [11, 12]. The formulas entered into Gazer's knowledge base are translated into "rewrite normal form" that is similar to the form knowledge takes in this dissertation although, in Gazer, the formulas are assumed to be nearly Horn. Gazer uses abstraction planning to plan the entire proof but the proof has to have a certain linearity, which obliterates completeness. Gazer has an advantage over this author's framework on problems that only require reasoning that has the nature of rewriting. However, in non-Horn theories, something stronger than Gazer is required.

Hähnle, Klingenbeck and Pape's A-ordered tableaux are another restriction to clausal tableaux that reduce the search space tremendously while retaining completeness [64, 62]. This

system make gains in efficiency by applying an order on the predicates and terms in the theory.

Frank Brown [39] used what was essentially an incomplete sequent-based prover with free variables to prove theorems in set theory. Brown's system used one particular theory (Quine's set theory [103]) primarily as rewrite rules. Brown's work was very closely related to Bledsoe's IMPLY prover [29] that also used knowledge essentially for rewriting.

Baumgartner has worked to allow provers to use certain types of first-order theories by means of theory reasoning [13]. A theory reasoning system needs a background reasoner for the theories of interest. They are particularly useful for Horn theories.

Boyer and Moore's proof checkers, which are experts at applying induction, use theories extensively. In this case, the user needs to supply a bit of information along with the statement of the lemma to help the prover know how to use the knowledge [37].

Larry Hines' *hyper-chaining* technique is another example of a system in which the user needs to attach extra information to the knowledge so that the prover will know how to use it [68].

The problem of adding equality inference rules to tableau systems is quite different from that of adding equality to resolution systems [17, 18, 55]. Most of the work published in this area has concentrated on complete procedures. In this dissertation, we have looked for some "minimal" set of inference rules for equality that encompass a useful area of mathematical reasoning without becoming intractable.

Many interactive theorem provers and proof checkers have been successful in proving many theorems in set theory. In this work we emphasized the automatic discovery of proofs in set theory. Related to this, there has been less done. Art Quaife used Gödel's first-order axioms (or a slight improvement on them which benefits systematization) to prove theorems in set theory [101]. The calculus introduced in Section 4.2 bypasses many of Gödel's axioms rendering them unneeded for most applications. However, the calculus described here does not construct sets well independently of hints from the knowledge base.

Woody Bledsoe initiated a tremendous amount of work on the automation of reasoning

in set theory. His Set-Var calculus was quite successful on many interesting problems [30]. Set-Var used a rule for constructing sets that was essentially equivalent to circumscription. Many of Bledsoe's collaborators (Plummer, Brown, etc.) also worked on related problems. Bailin and Barker-Plummer's Z-match took a very similar approach to Set-Var [10]. The present work does not construct classes in any interesting way as these systems do.

Frank Brown also worked on a system that proved theorems in set theory using some special inference rules [39]. These special inference rules encoded all of the axioms of Quine's set theory into rewrite rules. This method was also quite successful on many interesting problems. The present approach allows any set theory to be used rather than wiring one particular system into the calculus. The comprehension schema described in Section 4.2 is similar to one of the rewrite rules in Brown's work.

The use of breadth-first search for a tableau closure seems to be unique to this work. Although the possibility of such a thing has been known, the advantages and disadvantages have not been widely discussed nor has the problem of implementing strategies in the context of breadth-first unification.

7.2 Interface

Many systems have interfaces that are superior in many ways to the interface to IPR. What is more unusual about IPR is its ability to have such an interface so closely related to a strong automatic theorem proving calculus. In most systems, one of these properties excludes the other.

Some work has been done on the problem of having automated theorem proving systems read natural language [112]. However, most systems, even interactive systems, require the user to input the statements in a formal computer-oriented language. IPR is not unusual in this way.

In order to produce all of its output in natural language, IPR requires format strings to be associated with predicate and term definitions. Any system that translates formulas and terms into natural language must have this information given to it in some form [53].

In terms of the interaction, IPR is similar to interactive theorem proving systems [53]. Powerful automatic systems do not offer the non-expert user the opportunity to watch a proof in progress and change its direction. With few exceptions [74], automatic systems cannot produce a natural language description of a proof once it is finished.

Boyer, Moore and Kaufmann's NQTHM prover communicates with the user largely in English although formulas and terms in the formal language are not translated [38].

David McAllester developed a formal language that more closely resembles English [84, 87]. The proof system, Ontic, checks proofs in this language.

Chapter 8

Conclusion

> ...one could be too conservative in estimating the potentialities of machines in theorem proving.
>
> —Hao Wang. 1960 [116]

In this chapter, we summarize the new results presented in this dissertation and present a history of the author's development of the ideas behind IPR and behind the IPR program itself. Since there is plenty of room for improvement and many problems that remain unsolved, we also discuss some of these problems and some ideas for extending this work. We include in Section 8.5 some remarks about why IPR is successful on problems which are difficult for other provers.

One of the long-range goals is to solve the $n+1$ problem. That is, given the first n statements in some textbook, have the prover prove the $(n+1)^{\text{st}}$. By enabling the intelligent use of a large knowledge base, this procedure is an excellent step along with the great progress made by many others in the field. We discuss this challenge and some of what will be necessary to solve it in more generality in Section 8.3.

8.1 Summary

The IPR prover uses several novel logical rules and control ideas that contribute to its success.

The non-clausal form introduced in this work (first described in an early technical report [106]) has several advantages in both efficient reasoning and human interface (Section 2.3.)

The ε-rule is a generalized connection method for applying knowledge. It can be applied in both the clausal and non-clausal contexts. In the clausal context, completeness is retained even with the weak connection condition.

The non-clausal ε-rule was developed with the intention of developing a calculus for applying knowledge in a way that is easy to explain. The non-clausal form is also believed to be a benefit for efficient reasoning. Since the application of the ε-rule resembles the way a human explains the application of a theorem, it makes it easier (than in the case of resolution, for example) for the programmer to think about human-like strategies for selecting theorems to apply [98, 114].

The strategies mentioned in Section 3.3 include the "$n + m$" strategy that prefers to apply theorems that do not add too much complexity to what remains to the proof after the theorem application. Other strategies mentioned prefer to apply theorems that have as much in common as possible with the formulas and terms in the proof so far. Finally, the set-of-support strategy requires that a theorem be applied only when some predicate in the theorem is unifiable with some properly signed predicate in the proof so far.

Once the logical rules are described, the question of control arises. IPR implements several novel methods in controlling how rules are applied and how a proof is sought. One of the most important of these control issues is breadth-first unification. IPR searches for a unification that closes a tableau breadth-first. Breadth-first unification allows several other strategies, such as subgoal (or branch) selection [75], to be applied more easily than in the case of depth-first unification. Also, if it is done as described in Section 5.1.1, breadth-first unification allows theorems to be applied using substitutions that are incompatible with the proof and yet the proof can still be found in a proof-confluent way.

Another issue of control is the splicing of formulas onto a branch at the highest position possible. This saves space, usually makes the presentation of the proof more attractive and allows shorter proofs to be found in some cases (Section 5.1.)

Every aspect of the new calculus developed for IPR was conceived in order to facilitate human interface. IPR is able to explain the proofs it finds in English. Also, when a proof is

not successful, the user can navigate a tree of sequents (in English) and understand what has been done in the proof so far and what remains to be done. If there is some knowledge missing from the knowledge base, the user can often understand the incomplete proof well enough to tell what is missing.

The user can also direct the proof by giving commands about rules to apply. Thus the user can have full control over the construction of a proof. The user can also ask the prover for a list of options.

Chapter 6 contains a list of interesting examples that have been proved by IPR. One of those examples has not yet been proved by any other prover. Many of these examples were proved in the presence of knowledge that was related to the problem but not needed in the proof. The presence of unneeded knowledge makes these problems much more difficult.

In order to allow the user to get into advanced mathematics more easily, IPR also implements the novel handling of the comprehension schema and the definite descriptor (ι-operator.)

In Chapter 6 and in Section 4.2.6, examples of theorems in set theory and higher-order logic are given. IPR uses a novel method for proving higher-order theorems in set theory. The method uses the comprehension schema and seems to give IPR a great advantage over provers using other methods on many problems. Several theorems that are used as examples to show the strength of other systems are proved without search using this tractable comprehension schema method.

The result of all of these discoveries takes the technology for developing really useful and usable automated theorem proving applications a long way. IPR is both a strong automated theorem proving program that explains its proofs in English and an easy to use interactive prover that explains proofs in progress in a comprehensible way.

8.2 A Brief History of IPR

In 1993, I began by writing a prover for first-order logic. I was initially influenced by experience with Bledsoe's IMPLY [29] and McAllester's Ontic [84]. My plan was to implement something

that would be as sensible and easy to use as these provers except that it would be a theorem prover (like IMPLY) instead of a proof checker (like Ontic.) I wanted the program to write a readable proof rather than having the human write a readable proof to be checked by the computer.

The earliest implementation of IPR was a sequent-based prover. I read Smullyan's *First-Order Logic* [113] and followed his directions. Then I began to make improvements, reinventing everything from free-variable semantic tableaux [54] to Brown's depth-first non-backtracking unification strategy [39].

I was also experimenting with the interface. I strove to have the prover present proofs in progress in a way that was easy to understand. I implemented the tree navigation code for the interface very early in the process and experimented with various conventions for displaying Skolem terms.

In July 1993, I visited McAllester at MIT and Farmer, Guttman and Thayer at the MITRE Corp. Farmer, *et. al.* had implemented IMPS [52], a sequent based proof checker for higher-order logic. I was very impressed by their interface and began to try to make my system as easy to use.

In late 1993 and early 1994, I implemented the congruence closure technique [85, 105] into the unifier. It was at that time that the more complex equality problems were solved by the prover.

At the same time, I experimented with the sequent calculus and its variations including "lemmaizing" [5], mixed universal and rigid variables [18], various δ-rules [8], equality methods [18, 85] and unification strategies. In those early days, the prover was tested on Pelletier's problems [97] and was quite successful, although not enough to be competitive with the provers that win the international contests.

In the Spring of 1994, I worked on adding rules for handling set theory. This included the conservative comprehension schema and other simple rules for drawing conclusions about classes as described in Section 4.2.

In the Summer of 1994, inspired by the $n+1$ challenge, I got more serious about using a knowledge base automatically. At that time, I developed the non-clausal ε-rule and restrictions described in Chapters 2 and 3. The first theorems proved by IPR using knowledge were from set theory.

During this period from 1993 through 1994, I was also developing the interface. I programmed the part of the user interface that allows the user to give commands. The user was able to use equalities in the hypotheses to make arbitrary substitutions. The user was able to tell the program to apply any of the logical rules to an appropriate formula. Once the ε-rule was invented, I also implemented the code for translating formulas into English. Until that point, only the logical operators were in English. The code for producing English proofs was also developed during this time.

From August 1994 through May 1996 I worked an unrelated project with Benjamin Kuipers [111].[1] During that time, the only work I got done related to IPR was light writing and editing of the technical reports describing the non-clausal ε-rule, the associated strategies, the comprehension schema, the congruence closure and other basic ideas behind IPR [106, 107].

In the Summer of 1996, I began working on the IPR code again in earnest. At this point, I was most interested in its ability to prove theorems in advanced theories using relatively large knowledge bases. I wanted to test the strength of my non-clausal ε-rule and the associated strategies. The main goal of my work that summer was to get IPR to succeed at Example 6.1. In the middle of August 1996, IPR proved this example.

8.3 The Theorem $n+1$ Challenge

There have been many factors motivating the work on IPR. One ambitious challenge has been proposed: enter the first n statements from a textbook in mathematics with as little extra information as possible and let the prover prove the next statement. This challenge was in

[1] This project had begun in the Fall of 1992 but I worked on it to the exclusion of all else only during the time mentioned. Because of this early association with Kuipers, the early versions of IPR used his Algernon system to store and access the knowledge [47].

place at least as early as Hao Wang's work in the late 1950s [116]. This has been called the $n+1$ problem or the theorem $n+1$ challenge.

John Kelley's text, *General Topology* [77], seems to be an excellent target for this test since it is very thorough, includes the axioms of set theory on which the system may be built, is apparently based on a system of axioms including only one proper schema (the easily-implemented comprehension schema) and contains many different types of reasoning. The version of the comprehension schema given by Kelley is implemented in IPR except that IPR also allows for ur-elements.

A prover that could solve the $n+1$ problem for all n would certainly be able to go further and prove open questions in the area of mathematics covered by the text. A prover that does this well will be very useful to mathematicians, physicists and others who could benefit from an assistant reasoner. Therefore, I discuss here what seem to be the barriers that remain between our current technology and the solution to the $n+1$ challenge.

Fetching. Any prover attempting to solve this problem will have to use a large knowledge base. This is true even if the challenge is restated so that only statements from "relevant" sections are in the knowledge base. The IPR framework is a step toward solving the fetching problem. IPR is particularly weak when a proof requires a good deal of rewriting. Barker-Plummer's prover, Gazer, implements a fetcher that is stronger on rewriting problems [11] but more limited in non-Horn theories. The Gazing algorithm accomplishes this by planning the entire (if linear) proof before applying any theorem. Barker-Plummer's implementation limited the application of theorems to exclude branching altogether and limited the formulas input to the knowledge base to be nearly Horn. A combination of the ideas in Gazer and IPR seems very promising. As in Gazing, proofs could be planned but the knowledge base would have the more general form as in IPR and the plans would be allowed to branch. In order to keep the complexity down, the plans would be formed in a more intelligent manner using IPR's strategies.

Other promising technology exists for helping to solve the fetching problem. For

example, Hähnle's A-ordered tableaux make tremendous restrictions to the way knowledge can be applied [64].

Rewriting. This problem has been solved to a great extent in many systems. Indeed, even pure mathematicians frequently use softwares, such as Mathematica, that apply rewriting techniques to help them prove theorems. The solution to the fetching problem mentioned above would go quite far in the direction of controlling rewriting but some proofs will require some of the more powerful rewriting techniques that are used by other systems [118, 122, 123]. This would be even more important for texts in more algebraic fields.

The current framework does not control rewriting well at all. There are many existing provers that easily outperform it on problems that are Horn or otherwise related to rewriting. There is no reason to believe that those methods will be impossible to combine with this framework, thus creating a prover that will be able to solve the $n + 1$ problem for more and more n.

Set instantiation. In proofs such as Kelley's proof of Tychonoff's theorem, it is necessary to construct a set that has certain properties. IPR can prove this theorem if we cheat by putting an extra lemma into the knowledge base that contains the set needed in the proof or if we interact once with the prover to tell it what that set must be. Methods such as higher-order unification used in Andrews' TPS prover [3] would need to be incorporated into a prover that we want to discover this kind of proof without help. This is an area where much more work is needed.

Finding examples and counter-examples. IPR is able to find examples and counter-examples as long as the needed example is mentioned in some lemma in the knowledge base. For instance, IPR may be able to prove that there is a regular topological space that is not normal if, for example, (1) there is a theorem in the knowledge base that mentions that a certain space is regular *or* not normal and (2) that space happens to be regular *and* not

normal. Without this kind of knowledge some very strong higher-order reasoning seems to be needed to solve this kind of problem.

Induction. There is not as much proof by induction in Kelley's book as there would be in a text on number theory or algebra. It appears that perhaps all of the induction in this text is within the reach of a good fetcher assuming that the principle of mathematical induction, as Kelley states it, is included in the knowledge base. Kelley's principle of mathematical induction is a first-order formula that simply states that every inductive subset of ω is identical to ω. If we were to choose a number theory text that used a schema for its induction principle then the problem of finding proofs by induction becomes more difficult. For information on successful provers in this area, reference the Boyer-Moore provers [36] and the rippling provers [41].

Matthias Baaz *et. al.* recently demonstrated an excellent way of implementing induction and related inference rules in a tableau based prover [7].

Complexity. Finally, there is the simple fact that some proofs are just too long and complex to be handled by existing methods with existing commonly used hardware. Proof planning and analogy could help this problem by keeping the search space smaller.

There is another method, proposed long ago, for handling complex mathematical proofs. That is, in Hao Wang's words, the formation of "interlocked hierarchies of methods, and ...a complex web of clearly understood, definite and deterministic algorithms..." [116]. What is intended here is the combination of special methods, many of which will be decision procedures, with a program that assigns each part of the proof to its appropriate method. In the words of Woody Bledsoe,

> The stumbling block to prover designers is the desire for a "magical" solution, a "lodestone," a simple algorithm that will search out and eventually find a proof. But alas, it requires more complicated mechanisms to do complicated reasoning; the human prover uses a whole collection of methods of different sorts.
>
> —Woody Bledsoe [33]

To my knowledge, such a project has not been undertaken on a scale large enough to attempt the $n+1$ problem. However, this idea seems as promising today as it did 37 years ago.

8.4 Imitating Human Techniques.

Some people give me credit for imitating human reasoning techniques with the calculi I have presented here. I do not deny this except to say that no one really knows how a human proves theorems. It might be said that the calculus presented here resembles loosely what happens on a conscious level when a human is doing certain types of reasoning. The goal has been more to explain computer reasoning to humans than to imitate the human psyche.

Others criticize this resemblance as if it were a weakness in the calculus. I do not understand this criticism. These detractors say something like the following.[2]

> Since the characteristics of computers and of people are apparently quite different, no need exists for emulating the reasoning a person might employ, for the objective is to design and implement powerful and effective procedures to be used in a computer program that reasons; a study of how people solve problems, however, might lead to important breakthroughs.
>
> —Larry Wos [120]

The famous quotation attributed to Edsgar Dijkstra is also used: to ask whether a computer can be intelligent is like asking whether a submarine can swim. The point is that computers and humans presumably operate differently. We should expect that computers will do some things better than people and vice-versa. I have no argument against any of these remarks. I quite agree that "the objective is to design and implement powerful and effective procedures to be used in a computer program that reasons." It seems that the point being made here is that this objective should be obtained by any means necessary. In other words:

> if something works well, use it *regardless of whether or not* it resembles what we think of as human reasoning.

[2] I do not mean to say that Larry Wos is one of the detractors. This quote sums up rather nicely the argument that others use against the imitation of human techniques.

All of this I accept.

However, it absolutely does not follow that the imitation of human techniques is excluded. In fact the detractors contradict the very statement that they imagine is supporting them. They essentially say the following:

> if something works well, use it *unless* is resembles what we think of as human reasoning.

This is not only a non sequiter, but in contradiction with the previous statement.

Surprisingly, the first statement, which is a paraphrase of the words of Larry Wos, supports very strongly the promise of the idea of the imitation of human techniques. Human techniques, whatever they are, obviously work better than current computer techniques on many hard problems.

I am not arguing here that my techniques mirror human reasoning in any deep way. I am only arguing that if they do mirror human reasoning in any way at all, this is not something that anyone should be embarrassed about or surprised by.

8.5 Synergistic Effect

The success of IPR on examples such as Example 6.1 is due to the combination of four factors: the ε-rule, the strategies mentioned in Section 3.3, breadth-first unification and the use of the non-clausal form.

The strength of breadth-first unification is that it tries all possibilities without backtracking. It is not used with other tableau systems because it is very expensive. It must try around n^k combinations of closing substitutions where k is the number of branches and n is the number of closing substitutions per branch. The strategies applied with the ε-rule reduce the number of branches tremendously and thereby make breadth-first unification viable.

The ε-rule allows any number of connected formulas (the formulas in $U' \cup V'$) and those formulas are not added to the tableau. Therefore, it produces tableaux with fewer branches than other tableau expansion rules. Thus, it demonstrates its strength when combined with

breadth-first unification while other tableau expansion rules (it is widely believed) do not benefit from the use of breadth-first unification.

These strategies (particularly the $n+m$ strategy) cause IPR to prefer to apply knowledge which will keep the number of branches low. Thus, the combination of the strategies and the ε-rule make breadth-first unification a more reasonable unification strategy. Even the combination of the ε-rule with breadth-first unification is not able to prove Example 6.1 in a reasonable amount of time without the application of the strategies for selecting knowledge as described in Section 3.3.

If the ε-rule, the strategies or breadth-first unification were used alone, or together with other widely-used techniques, none of them would significantly increase the power of the associated theorem proving system. However, their combination is quite powerful.

The last factor involved in the success of IPR is the use of the non-clausal form described in Section 2.3.2. With the non-clausal form there are fewer items in the knowledge base (Section 2.3.1.) Since each of the items in the knowledge base must be searched for connections into the tableau, the benefit of having fewer items is clear. Furthermore, the items in the knowledge base are at a slightly higher level. This allows matches to be made earlier in the proof search process. In typical problems, the advantages of the use of the non-clausal form are relatively small compared to the advantages gained by the other novelties mentioned above.

In short, the strength of the prover on examples such as Example 6.1 comes primarily from the combination of the use of the ε-rule, the strategies for keeping the complexity of the tableau down and the use of breadth-first unification. The use of the non-clausal form also contributes to efficiency.

8.6 Future Work

The calculus presented here is compatible with many of the refinements studied in the literature of clausal tableaux. It would be interesing to explore the benefits of adding some of these refinements. For example, the problem of adding a regularity condition or dynamic

lemma generation in the context of breadth-first unification might bring up some interesting theoretical and implementational questions.

There is a possibility to obtain some of the advantages to using clausal form while maintaining some of the advantages of the non-clausal form. One such possibility is to store the knowledge internally in clausal form and display it to the user in the non-clausal form. This might bring other problems into the interface, such as introducing Skolem functions in a way that might surprise the user but they could probably be handled without too much difficulty. The problem of keeping new Skolem functions distinct from the Skolem functions in the knowledge base would not be difficult to solve.

It is likely that more work on the ranking that IPR applies to theorem applications when it is fetching could be improved. Other ideas related to the ones explained in Section 3.3 could be implemented and tested on harder problems.

Some techniques for selecting input clauses recently developed for clausal tableaux may be compatible with the goals of this project. Hähnle's A-ordered tableau ideas [62] may augment IPR's selection criteria well. The problem faced here is that the user (or perhaps the program) needs to decide on a selection function or an ordering on terms and predicates. It is this kind of decision that we would like to keep away from the non-expert user.

There is also the possibility of an extension or generalization of the planning mechanism used in Gazer [11]. The techniques used by IPR for knowledge selection are local whereas if the proof is planned, then we have a global view of the proof that should help in the decision making.

Furthermore, ideas from learning and training software could be used to train IPR to be more intelligent in its selection of theorems.[3]

Brown's Rule is the only *substitution* of equals that IPR currently applies automatically. Further experimentation may recommend other rules in the future, especially rules

[3] This idea was suggested by John Case of the University of Delaware.

related to higher-order logic (Section 4.2.4.) It should also be rewarding to research the compatibility of complete methods for handling equality with the current calculus and goals.

Although it takes away from the general-purpose nature of IPR, the method used by Frank Brown for handling set theory seems promising [39]. The procedure I envision is to write tableau expansion and branch closure rules for each axiom of, say, Kelley's set theory. A related problem is to try to have classes be constructed intelligently as is needed for the satisfactory solution of the problem posed by Wos to test a system claiming to solve his eighth research problem [120].

Experimentation has convinced me that control issues are an extremely important factor contributing to the success or failure of a program using any given logical calculus. In what order should rules be applied? How often should equality, breadth-first unification and trivial ε-rule applications be applied and how aggressively? These and many other similar questions are typically left open by the definition of a calculus but their answer makes an enormous difference in the success of an implementation.

Many systems allow (or require) the user to include some strategic information along with the knowledge. This information is used to help the program know how or when to use the knowledge. It seems that it would not be difficult for a non-expert user to have the ability to know, at some level, how certain knowledge should or should not be applied. A system for allowing the user to inform the program of such preferences ought to improve performance.

The IPR prover has such a natural interface that it is possible to write educational software based on the prover. A course could be designed to teach proof skills and the students could use a limited version of IPR to prove theorems. For example, a limited version of IPR could be written that gives hints, recommendations or just a list of possibilities instead of taking steps automatically. The student would have to decide which theorem to apply and how to apply it. The student would have to decide how to instantiate variables in a proof. The prover could give a list of possible instantiations or a list of possible theorems to apply. The desired difficulty of the course would determine how much help the prover would offer.

Various aspects of the interface need to be improved and augmented. It would be easy

to give the user the ability to select from the possible ways a branch can be closed. When the user makes this choice, the prover would apply that substitution across the tree, closing many branches and cutting others off using the condense algorithm [94]. It would also be easy to program IPR to show the user all the top-ranked options for applying theorems and allow the user to make a selection. All of the information needed to do this is stored on the tableau. All that is lacking is the interface to the user.

It would be nice to have time to try more examples from advanced mathematics. It takes time to try example challenge problems because the knowledge all has to be entered and checked. The examples given in this dissertation only give an idea of the kind of reasoning that the prover can do.

The future of automated theorem proving looks promising. Consider a comparison between the future of proof checking to the future of automated proof discovery on hard examples. It will always take several hours, days or weeks for an expert at a proof checking system to program an acceptable proof of a difficult example. The amount of time this takes is not decreasing. The advantage to proof checking software is that, with enough time, the user can check the proof of almost any theorem for which a proof is known. However, the time it takes for automated theorem proving software to prove difficult examples is always decreasing and the difficulty of the problems that are being solved by these systems is always increasing. There is no reason to believe that this trend is going to end in the near future. Unless there is some as yet invisible bound on this sequence, automated theorem proving programs will be able to prove almost any theorem that a person would ever care to write out in a checkable form in less time than it takes the person to write it. It seems more reasonable to expect that the technology that lies between these extremes will prove most successful. That is to say that interactive theorem provers, with very strong underlying ATP engines, will be able to outperform both pure ATP and pure proof checking systems with some little bit of guidance from a user.

We close with a challenge that seems to be just out of reach of the IPR prover. Here

is the statement of the theorem.

$$\begin{gathered}(\text{continuous-from-to}(\phi, S, T) \wedge \text{Hausdorff}(T)) \supset \\ \text{closed-in}(\Gamma_\phi, S \times_\tau T)\end{gathered}$$

The proof should be found in the theory given in Figure 8.1. The theory is not minimal for the problem, which makes it harder. Here $f(x)$ is an abbreviation for, say, $\text{apply}(f, x)$ and Γ_f stands for the graph of f as a set.

the definition of closed
$$\text{closed-in}(A, X) \leftrightarrow$$
$$(\forall y)((y \in \text{top-to-class}(X) \wedge y \notin A) \supset$$
$$(\exists G)(y \in G \wedge \text{open-in}(G, X) \wedge \text{disjoint}(G, A)))$$

the definition of Hausdorff:
$$(\forall X)(\text{Hausdorff}(X) \leftrightarrow (\forall A)(\forall B)((A \in \text{top-to-class}(X) \wedge B \in \text{top-to-class}(X) \wedge$$
$$A \neq B) \supset (\exists G1)(\exists G2)(\text{open-in}(G1, X) \wedge \text{open-in}(G2, X) \wedge A \in G1 \wedge$$
$$B \in G2 \wedge \text{disjoint}(G1, G2))))$$

the definition of a continuous function
$$\text{continuous-from-to}(f, X, Y) \leftrightarrow$$
$$(\text{function-from-to}(f, \text{top-to-class}(X), \text{top-to-class}(Y)) \wedge$$
$$(\forall G)(\text{open-in}(G, Y) \supset \text{open-in}(f^{-1}(G), X)))$$

the definition of product topology:
$$(\text{open-in}(A, X) \wedge \text{open-in}(B, Y)) \supset$$
$$\text{open-in}(A \times B, X \times_\tau Y)$$

part of the definition of product of topological spaces:
$$X \in \text{top-to-class}(S \times_\tau T) \supset$$
$$(\exists A)(\exists B)(A \in \text{top-to-class}(S) \wedge B \in \text{top-to-class}(T) \wedge X = \langle A, B \rangle)$$

the definition of product:
$$X \in S \times T \leftrightarrow (\exists A)(\exists B)(A \in S \wedge B \in T \wedge X = \langle A, B \rangle)$$

the definition of disjoint:
$$(\forall A)(\forall B)(\text{disjoint}(A, B) \leftrightarrow \neg(\exists Y)(Y \in A \wedge Y \in B))$$

a fact about inverse images
$$x \in f^{-1}(A) \leftrightarrow f(x) \in A$$

a fact about graphs
$$\langle a, b \rangle \in \Gamma_f \leftrightarrow b = f(a)$$

a basic fact about functions
$$(\text{function-from-to}(f, A, B) \wedge a \in A) \supset f(x) \in B$$

Figure 8.1: The knowledge for the challenge.

Bibliography

[1] R. Abraham, J. E. Marsden, and T. Ratiu. *Manifolds, Tensor Analysis, and Applications*. Springer-Verlag, second edition, 1988.

[2] Peter B. Andrews. *An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof*. Academic Press, 1986.

[3] Peter B. Andrews, Matthew Bishop, Sunil Issar, Dan Nesmith, Frank Pfenning, and Hongwei Xi. TPS: A theorem proving system for classical type theory. Technical Report 94–166, Department of Mathematics, Carnegie Mellon University, 1994.

[4] Peter B. Andrews, Dale A. Miller, Eve Longini Cohen, and Frank Pfenning. Automating higher-order logic. In W. W. Bledsoe and D. W. Loveland, editors, *Automatic Theorem Proving: After 25 Years*, volume 29 of *Contemporary Mathematics*, pages 169–192. American Mathematical Society, 1984.

[5] O. W. Astrachan and M. E. Stickel. Caching and lemmaizing in model elimination theorem provers. In *Proc. 11th Conference on Automated Deduction*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 224–238. Springer Verlag, 1992.

[6] Owen Astrachan. METEOR: Exploring model elimination theorem proving. *Journal of Automated Reasoning*, 13(3):283–296, 1994.

[7] Matthias Baaz, Uwe Egly, and Christian G. Fermüller. Lean induction principles for tableaux. In Didier Galmiche, editor, *Automated Reasoning with Analytic Tableaux and Related Methods*, volume 1227 of *Lecture Notes in Artificial Intelligence*, pages 62–75. Springer-Verlag, May 1997.

[8] Matthias Baaz and Christian G. Fermüller. Non-elementary speedups between different versions of tableaux. In Peter Baumgartner, Reiner Hähnle, and Joachim Posegga, editors, *Proc. 4th Workshop on Deduction with Tableaux and Related Methods, St. Goar, Germany*, LNCS 918, pages 217–230. Springer-Verlag, 1995.

[9] L. Bachmair and N. Dershowitz. Completion for rewriting modulo a congruence. *Theoretical Computer Science*, 67(2 & 3):173–201, 1989.

[10] Sidney C. Bailin and Dave Barker-Plummer. Z-match: An inference rule for incrementally constructing set instantiations. *Journal of Automated Reasoning*, 11(3):391–428, 1993.

[11] Dave Barker-Plummer. Gazing: An approach to the problem of definition and lemma use. *Journal of Automated Reasoning*, 8(3):311–344, 1992.

[12] Dave Barker-Plummer and Alex Rothenberg. The Gazer theorem prover. In D. Kapur, editor, *Proceedings of the Eleventh International Conference on Automated Deduction*, volume 607 of *Lecture Notes in Computer Science*, pages 726–730. Springer-Verlag, 1992.

[13] P. Baumgartner. A model elimination calculus with built-in theories. In Fronhöfer, Hähnle, and Käufl, editors, *Theorem Proving with Analytic Tableaux and Related Methods*, 1992.

[14] P. Baumgartner and U. Furbach. PROTEIN: A *PRO*ver with a *T*heory *E*xtension *I*nterface. In A. Bundy, editor, *Automated Deduction – CADE-12*, volume 814 of *Lecture Notes in Artificial Intelligence*, pages 769–773. Springer, 1994. Available in the WWW, URL: `http://www.uni-koblenz.de/ag-ki/Systems/PROTEIN/`.

[15] P. Baumgartner, U. Furbach, and I. Niemelä. Hyper Tableaux. In *JELIA 96*. European Workshop on Logic in AI, Springer, LNCS, 1996.

[16] B. Beckert and J. Posegga. leanT^AP: Lean, tableau-based deduction. *Journal of Automated Reasoning*, 15(3):339–358, 1995.

[17] Bernhard Beckert. Adding equality to semantic tableaux. In Krysia Broda, Marcello D'Agostino, Rajeev Goré, Rob Johnson, and Steve Reeves, editors, *Proc. 3rd Workshop on Theorem Proving with Analytic Tableaux and Related Methods*, pages 29–42, April 1994.

[18] Bernhard Beckert. A completion-based method for mixed universal and rigid E-unification. In Alan Bundy, editor, *Proc. 12th Conference on Automated Deduction CADE, Nancy/France*, volume 814 of *Lecture Notes in Artificial Intelligence*, pages 678–692. Springer-Verlag, 1994.

[19] Bernhard Beckert, Stefan Gerberding, Reiner Hähnle, and Werner Kernig. The tableau-based theorem prover ${}_3T^AP$ for multiple-valued logics. In *Proceedings, 11th International Conference on Automated Deduction (CADE), Saratoga Springs, NY*, LNCS 607. Springer, 1992.

[20] Bernhard Beckert, Reiner Hähnle, and Peter H. Schmitt. The *even more* liberalized δ-rule in free variable semantic tableaux. In Georg Gottlob, Alexander Leitsch, and Daniele Mundici, editors, *Proceedings of the third Kurt Gödel Colloquium KGC'93, Brno, Czech Republic*, volume 713 of *Lecture Notes in Computer Science*, pages 108–119. Springer-Verlag, August 1993.

[21] Paul Bernays. *Axiomatic Set Theory: with a Historical Introduction by Abraham A. Fraenkel.* Dover Publications, 1991. This Dover edition, first published in 1991, is an unabridged and unaltered republication of the second edition (1968) of the work first published by the North-Holland Publishing Company, Amsterdam, in 1958 in their series *Studies in Logic and The Foundations of Mathematics.*

[22] E. W. Beth. *The Foundations of Mathematics.* North Holland, 1959.

[23] E. W. Beth. On machines which prove theorems. In Jörg Siekmann and Graham Wrightson, editors, *Automation of Reasoning*, volume 1, pages 79–90. Springer-Verlag, 1983. Reprinted from *Simon Stevin Wis-en Naturkundig Tijdschrift*, Vol. 32, pages 49–60, 1958.

[24] Richard L. Bishop and Samuel I. Goldberg. *Tensor Analysis on Manifolds.* Dover Publications, 1980. This Dover edition, first published in 1980, is an unabridged and corrected republication of the work originally published by The Macmillan Company in 1968.

[25] W. W. Bledsoe and L. J. Henschen. What is automated theorem proving? *Journal of Automated Reasoning*, 1(1), 1985.

[26] Woodrow W. Bledsoe. Non-resolution theorem proving. *Artificial Intelligence*, 9:1–35, 1977.

[27] Woodrow W. Bledsoe. I had a dream: AAAI Presidential address, 19 August 1985. *The AI Magazine*, pages 57–61, 1985.

[28] Woodrow W. Bledsoe. Interactive proof presentation. In J. L. Sassez and Gordon Plotkin, editors, *Computational Logic*, pages 136–165. MIT Press, 1991.

[29] Woodrow W. Bledsoe and Peter Bruell. A man-machine theorem-proving system. *Artificial Intelligence*, 5:51–72, 1974.

[30] Woodrow W. Bledsoe and Guohui Feng. Set-Var. *Journal of Automated Reasoning*, 11(3):293–314, 1993.

[31] Woodrow W. Bledsoe and Larry M. Hines. Variable elimination and chaining in a resolution-based prover for inequalities. In W. Bibel and R. Kowalski, editors, *Proceedings of the Fifth Conference on Automated Deduction*, volume 87 of *Lecture Notes in Computer Science*, pages 70–87. Springer Verlag, 1980.

[32] Woody Bledsoe. Splitting and reduction heuristics in automatic theorem proving. *Artificial Intelligence*, 2:55–77, 1971. also in *Automation Of Reasoning - Classical Papers On Computational Logic, Vol. II, 1967-1970*, ed. J. Siekmann And G. Wrightson, Springer-Verlag, Berlin, 1983, pp. 508-530.

[33] Woody Bledsoe. Some thoughts on proof discovery. Technical report, Microelectronics and Computer Technology Corporation, June 1986.

[34] George Boole. *An Investigation of the Laws of Thought.* Walton, London, 1854. Reprinted by Dover Books, New York, 1954.

[35] S. Bose, E. M. Clarke, D. E. Long, and S. Michaylov. Parthenon: a parallel theorem prover for non-horn clauses. *Journal of Automated Reasoning*, 8:153–181, 1992.

[36] Robert S. Boyer, M. Kaufmann, and J Strother Moore. The Boyer-Moore theorem prover and its interactive enhancement. *Computers and Mathematics with Applications*, 29(2):27–62, 1995.

[37] Robert S. Boyer and J. Strother Moore. *A Computational Logic Handbook*, volume 23 of *Perspectives in Computing.* Academic Press, 1988.

[38] Robert S. Boyer and J. Strother Moore. A theorem prover for a computational logic. In M. E. Stickel, editor, *Proceedings of the Tenth International Conference on Automated Deduction*, volume 449 of *Lecture Notes in Computer Science*, pages 1–15. Springer Verlag, 1990.

[39] Frank M. Brown. Towards the automation of set theory and its logic. *Artificial Intelligence*, 10(3):281–316, 1978.

[40] Bruno Buchberger. Natural language proof in nested cells representation. In X. Huang, J. Pelletier, F. Pfenning, and J. Siekmann, editors, *First International Workshop on Proof Transformation and Presentation*, April 1997.

[41] A. Bundy, A. Stevens, F. van Harmelen, A. Ireland, and A. Smaill. Rippling: A heuristic for guiding inductive proofs. *Artificial Intelligence*, 62:185–253, 1993.

[42] Alan Bundy. *The Computer Modelling of Mathematical Reasoning.* Academic Press, 1983.

[43] D. Cantone, A. Ferro, and E. G. Omodeo. *Computable Set Theory*, volume 1. Oxford University Press, 1989.

[44] Lewis Carroll. *Through the Looking Glass: And What Alice Found There.* Puffin Classics. Puffin, 1996.

[45] Chin-Liang Chang and Richard Char-Tung Lee. *Symbolic Logic and Mechanical Theorem Proving.* Academic Press, London, 1973.

[46] Shang-Ching Chou. *Mechanical Geometry Theorem Proving.* D. Reidel Publishing, 1988.

[47] J. M. Crawford and B. J. Kuipers. Algernon – a tractable system for knowledge representation. *SIGART Bulletin*, 2(3):35–44, June 1991.

[48] B. I. Dahn, J. Gehne, T. Honigmann, L. Walther, and A. Wolf. Integrating logical functions with ILF. Technical report, Institut für Mathematik der Humboldt-Universität zu Berlin, 1994.

[49] Martin Davis. A computer program for Presburger's algorithm. In *Summer Inst. for Symbolic Logic*, pages 215–233. Cornell Press, 1957.

[50] Martin Davis. The prehistory and early history of automated deduction. In Jörg Siekmann and Graham Wrightson, editors, *Automation of Reasoning*, volume 1, pages 1–28. Springer-Verlag, 1983.

[51] Nachum Dershowitz and Jean-Pierre Jounnaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, pages 243–320. Elsevier Science Pub. B. V./MIT Press, 1990.

[52] W. M. Farmer, J. D. Guttman, and F. J. Thayer. IMPS: System description. In D. Kapur, editor, *Proceedings of the Eleventh International Conference on Automated Deduction*, volume 607 of *Lecture Notes in Computer Science*, pages 701–705. Springer-Verlag, 1992.

[53] William M. Farmer, Joshua D. Guttman, and F. Javier Thayer. IMPS: An interactive mathematical proof system. *Journal of Automated Reasoning*, 11(2):213–248, August 1993.

[54] Melvin C. Fitting. *First-Order Logic and Automated Theorem Proving.* Springer-Verlag, New York, second edition, 1996.

[55] J. Gallier, W. Snyder, P. Narendran, and D. Plaisted. Rigid E-unification is NP-complete. In *Proc. Logic in Computer Science LICS*, pages 218–227. IEEE Computer Society Press, 1988.

[56] H. Gelernter, J. R. Hansen, and D. W. Loveland. Empirical explorations of the geometry-theorem proving machine. In Jörg Siekmann and Graham Wrightson, editors, *Automation of Reasoning*, volume 1, pages 99–122. Springer-Verlag, 1983. Reprinted from *Proc. West. Joint Comp. Conf.*, May 1960, pages 143–147.

[57] Gerhard Gentzen. Investigation into logical deduction. In M. E. Szabo, editor, *The Collected Papers of Gerhard Gentzen*, pages 68–131. North-Holland Publishing Co., 1969.

[58] Robert Givan and David McAllester. New results on local inference relations. In *Principles of Knolwedge Representation and Reasoning: Proceedings of the Third International Conference*, pages 403–412. Morgan Kaufman Press, October 1992.

[59] Robert Givan, David McAllester, and Sameer Shalaby. Natural language based inference procedures applied to Schubert's steamroller. In *Proc. 9th National Conf. on Artificial Intelligence (AAAI-91)*, pages 915–920. Morgan Kaufmann Publishers, July 1991.

[60] Kurt Gödel. *The Consistency of the Axiom of Choice and the Generalized Continuum Hypothesis with the Axioms of Set Theory.* Princeton University Press, 1940.

[61] M. Gordon. HOL: A proof generating system for higher-order logic. In G. Birtwistle and P. A. Subrahmanyam, editors, *Current Trends in Hardware Verification and Automated Theorem Proving*, pages 77–128. Springer-Verlag, 1989.

[62] Reiner Hähnle and Stefan Klingenbeck. A-ordered tableaux. *Journal of Logic and Computation*, 1996, to appear. Available as Technical Report 26/95 from University of Karlsruhe, Department of Computer Science via anonymous ftp to `ftp.ira.uka.de` under `pub/uni-karlsruhe/papers/techreports/1995/1995-26.ps.gz`.

[63] Reiner Hähnle, Neil Murray, and Erik Rosenthal. Completeness for linear regular negation normal form inference systems. In Zbigniew Ras, editor, *Proc. International Symposium on Methodologies for Intelligent Systems, Charlotte, North Carolina*, Lecture Notes in Artificial Intelligence. Springer Verlag, 1997.

[64] Reiner Hähnle and Christian Pape. Ordered tableaux: Extensions and applications. In Didier Galmiche, editor, *Proc. International Conference on Automated Reasoning with Analytic Tableaux and Related Methods, Pont-à-Mousson, France*, volume 1227 of *Lecture Notes in Artificial Intelligence*, pages 173–187. Springer-Verlag, 1997.

[65] R. Hasegawa. MGTP: A model generation theorem prover—its advanced features and applications. In Didier Galmiche, editor, *Automated Reasoning with Analytic Tableaux and Related Methods*, volume 1227 of *Lecture Notes in AI*, pages 1–15. Springer-Verlag, May 1997.

[66] R. Hasegawa and Y. Shirai. Constraint propagation of CP and CMGTP: Experiments on quasigroup problems. In *Proceedings of the Twelfth International Conference on Automated Deduction*, 1994.

[67] D. Hilbert and W. Ackermann. *Principles of Mathematical Logic.* Chelsea, 1950. This is a translation of the second edition of *Grundzüge der Theoretischen Logik.*

[68] Larry M. Hines. Hyper-chaining and knowledge-based theorem proving. In E. Lusk and R. Overbeek, editors, *Proceedings of the Ninth International Conference on Automated Deduction*, pages 469–486. Spring-Verlag, 1988.

[69] Larry M. Hines. Str$\dot{+}$ve$\subset$: The Str$\dot{+}$ve-based subset prover. In Mark Stickel, editor, *Proceedings of the Tenth International Conference on Automated Deduction*, volume 449 of *Lecture Notes in Computer Science*, pages 193–206. Springer-Verlag, 1990.

[70] Larry M. Hines. The central variable strategy of Str$\dot{+}$ve. In D. Kapur, editor, *Proceedings of the Eleventh International Conference on Automated Deduction*, volume 607 of *Lecture Notes in Computer Science*, pages 35–49. Springer-Verlag, 1992.

[71] Jaakko Hintikka. Form and content in quantification theory. *Acta Philosohica Fennica*, 8:7–55, 1955.

[72] Jaakko Hintikka. *The Principles of Mathematics Revisited.* Cambridge University Press, 1996.

[73] Xiaorong Huang and Armin Fiedler. Presenting machine-found proofs. In *Proceedings of 13th International Conference on Automated Deduction*, 1996.

[74] Xiaorong Huang, Manfred Kerber, Michael Kohlhase, Erica Melis, Dan Nesmith, Jörn Richts, and Jörg Siekmann. Ω-MKRP: A proof development environment. In Alan Bundy, editor, *Proceedings of the 12th International Conference on Automated Deduction*, volume 814 of *Lecture Notes in Artificial Intelligence*, pages 788–792, Nancy, France, 1994. Springer-Verlag, Berlin, Germany.

[75] Ortrun Ibens and Reinholt Letz. Subgoal alternation in model elimination. In Didier Galmiche, editor, *Automated Reasoning with Analytic Tableaux and Related Methods*, volume 1227 of *Lecture Notes in AI*, pages 201–215. Springer-Verlag, May 1997.

[76] D. Kapur and N. Narendran. An equational approach to theorem proving in first-order predicate calculus. In *Proc. 9th Int. Joint Conf. on Artificial Intelligence (IJCAI-85)*, pages 1146–1153, 1985.

[77] John Kelley. *General Topology.* The University Series in Higher Mathematics. D. Van Nostrand Company, 1955.

[78] D. E. Knuth and P. B. Bendix. Simple word problems in universal algebras. In Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–267. Pergamon Press, 1970.

[79] Michael Kohlhase. Higher-order tableaux. submitted to the Tableau Workshop 1995, Koblenz.

[80] Leslie Lamport. *LaTeX: A Document Preparation System.* Addison-Wesley Publishing Company, second edition, 1994.

[81] R. Letz, K. Mayr, and C. Goller. Controlled integration of the cut rule into connection tableau calculi. *Journal of Automated Reasoning*, 13(3):297–338, December 1994.

[82] Reinhold Letz, Johann Schumann, Stephan Bayerl, and Wolfgang Bibel. SETHEO: A high-perfomance theorem prover. *Journal of Automated Reasoning*, 8(2):183–212, 1992.

[83] D.W. Loveland. A simplified format for the model elimination procedure. *Journal of the ACM*, 16(3):233–248, July 1969.

[84] David McAllester. *Ontic: A Knowledge Representation Language for Mathematics*. MIT Press, 1989.

[85] David McAllester. Grammar rewriting. In D. Kapur, editor, *Proceedings of the Eleventh International Conference on Automated Deduction*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 124–138. Springer-Verlag, June 1992.

[86] David McAllester and Robert Givan. Natural language syntax and first order inference. *Artificial Intelligence*, 56:1–20, 1992.

[87] David McAllester and Robert Givan. Taxonomic syntax for first order inference. *Journal of the ACM*, 40(2):246–283, April 1993.

[88] John McCarthy. Recursive functions of symbolic expressions and their computation by machine (Part I). *Communications of the ACM*, 3(3):184–195, April 1960.

[89] William McCune. Solution of the Robbins problem. *Journal of Automated Reasoning*. to appear.

[90] Elliott Mendelson. *Introduction of Mathematical Logic*. Mathematics Series. Wadsworth & Brooks/Cole, 3rd edition, 1987.

[91] James R. Munkres. *Topology: A First Course*. Prentice-Hall, Inc., 1975.

[92] G. Nelson and D. Oppen. Fast decision procedures based on congruence closure. *Journal of the ACM*, 27(2):356–364, 1980.

[93] A. Newell, J. C. Shaw, and H. A. Simon. Empirical explorations with the logic theory machine. In *Proc. West. Joint Comp. Conf.*, pages 218–239, 1957.

[94] F. Oppacher and E. Suen. HARP: A tableau-based theorem prover. *Journal of Automated Reasoning*, 4:69–100, 1988.

[95] S. Owen. *Analogy for Automated Reasoning*. Academic Press, Inc., 1990.

[96] Lawrence C. Paulson. Set theory for verification: I. from foundations to functions. *Journal of Automated Reasoning*, 11:353–389, 1993.

[97] Francis Jeffry Pelletier. Seventy-five problems for testing automatic theorem provers. *Journal of Automated Reasoning*, 2:191–216, 1986.

[98] G. Polya. *Mathematics and Plausible Reasoning*. Princeton University Press, 1954. Two volumes.

[99] Joachim Posegga. Deduction based on Shannon graphs. In *Proceedings GWAI-92, Bonn*, LNCS. Springer-Verlag, 1992.

[100] Dag Prawitz. An improved proof procedure. *Theoria*, 26:102–139, 1960. Reprinted in *Automation of Reasoning I* edited by Seikmann and Wrightson, pages 162–199, 1983, Springer-Verlag.

[101] Art Quaife. Automated deduction in von Neumann-Bernays-Gödel set theory. *Journal of Automated Reasoning*, 8(1):91–148, February 1992.

[102] Art Quaife. *Automated Development of Fundamental Mathematical Theories*, volume 2 of *Automated Reasoning Series*. Kluwer Academic Publishers, 1992.

[103] Willard Van Orman Quine. *Set Theory and its Logic*. The Belknap Press of Harvard University Press, revised edition, 1969.

[104] J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 1(12):23–41, 1965.

[105] Robert E. Shostak. An algorithm for reasoning about equality. *Comm. of the ACM*, 21(2):583–585, 1978.

[106] Benjamin Shults. The creation and use of a knowledge base of mathematical theorems and definitions. Technical Report ATP-127, The University of Texas at Austin, 1995.

[107] Benjamin Shults. A framework for the creation and use of a knowledge base of mathematical theorems and definitions. Technical Report ATP-127a, The University of Texas at Austin, 1995.

[108] Benjamin Shults. Challenge problems in first-order theories. *Association of Automated Reasoning Newsletter*, (34):5–8, October 1996.

[109] Benjamin Shults. A framework for using knowledge in tableau proofs. In Didier Galmiche, editor, *Automated Reasoning with Analytic Tableaux and Related Methods*, volume 1227 of *Lecture Notes in AI*, pages 328–342. Springer-Verlag, May 1997.

[110] Benjamin Shults. IPR and English proofs. In X. Huang, J. Pelletier, F. Pfenning, and J. Siekmann, editors, *First International Workshop on Proof Transformation and Presentation*, April 1997.

[111] Benjamin Shults and Benjamin Kuipers. Proving properties of continuous systems: qualitative simulation and temporal logic. *Artificial Intelligence*, 92:91–129, 1997.

[112] Donald Simon. Checking natural language proofs. In E. Lusk and R. Overbeek, editors, *Proceedings of the Ninth International Conference on Automated Deduction*, volume 310 of *Lecture Notes in Computer Science*, pages 141–150. Springer-Verlag, 1988.

[113] Raymond M. Smullyan. *First-Order Logic*. Dover Publications, New York, second corrected edition, 1995. First published 1968 by Springer-Verlag.

[114] Daniel Solow. *How to Read and Do Proofs*. John Wiley & Sons, Inc., 1982.

[115] Mark E. Stickel. A Prolog technology theorem prover: a new exposition and implementation in prolog. *Theoretical Computer Science*, 104:109–128, 1992.

[116] Hao Wang. Proving theorems by pattern recognition – I. In Jörg Siekmann and Graham Wrightson, editors, *Automation of Reasoning*, volume 1, pages 229–243. Springer-Verlag, 1983. Reprinted from *Communications of the Association for Computing Machinery*, Vol. 3, No. 4, April 1960.

[117] Hao Wang. Toward mechanical mathematics. In Jörg Siekmann and Graham Wrightson, editors, *Automation of Reasoning*, volume 1, pages 244–264. Springer-Verlag, 1983. Reprinted from IBM Journal of Research and Development, 4, 2–22, 1960.

[118] Stephen Wolfram. *The Mathematica Book*. Cambridge University Press, third edition, 1996.

[119] Larry Wos. Editorial: A journal is born. *Journal of Automated Reasoning*, 1(1), 1985.

[120] Larry Wos. *Automated Reasoning: 33 Basic Research Problems*. Prentice-Hall, Englewood Cliffs, 1988.

[121] Larry Wos. Automated reasoning: It cannot exist. Keynote Address at CADE 92, Saratoga Springs, June 1992.

[122] Larry Wos, Ross Overbeek, Ewing Lusk, and Jim Boyle. *Automated Reasoning: Introduction and Applications*. McGraw-Hill, Inc., second edition, 1992.

[123] Hantao Zhang. Herky: High performance rewriting in RRL. In D. Kapur, editor, *Proceedings of the Eleventh International Conference on Automated Deduction*, volume 607 of *Lecture Notes in Computer Science*, pages 696–700. Springer-Verlag, 1992.

[124] Hantao Zhang and Deepak Kapur. First-order theorem proving using conditional rewrite rules. In E. Lusk and R. Overbeek, editors, *Proceedings of the Ninth International Conference on Automated Deduction*, volume 310 of *Lecture Notes in Computer Science*, pages 1–20. Springer-Verlag, 1988.

Vita

Benjamin Price Shults was born in Austin, Texas on August 31, 1968, the son of Lynda Mae Shults and Fount Lee Shults. Mr. Shults attended high school at Honeoye Falls-Lima High School in Honeoye Falls, New York. There he concentrated academically on mathematics and physics and wrote a special report on Einstein's theories of relativity. He was also honored for his accomplishments on the swim team, soccer team and in musical drama. After completing his high school work in 1986, he entered Colgate University in Hamilton, New York. At Colgate, he pursued his interests in music, philosophy and physics while majoring in mathematics. He received a degree of Bachelor of Arts *magna cum lauda* with high honors in Mathematics from Colgate University in 1990. In September 1990 he entered the Graduate School of the University of Texas at Austin. His initial interests were in topology but he soon decided that it should be possible to program a computer to be helpful in proving theorems and joined Woody Bledsoe's vision. In October of 1992, he married Alice Elaine Neff. He also spent a couple of years while in graduate school working on a project with Benjamin Kuipers in which they developed a program which would check whether statements in temporal logic were true of the solutions to incompletely specified differential equations.

Permanent address: P.O. Box 753
Gambier, Ohio 43022

This dissertation was typeset with LaTeX[‡] by the author using the U.T. Austin Doctoral Dissertation Format - Version 1.1-a by Young. U. Ryu with some fixes by Miguel A. Lerma.

[‡]LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.