The Creation and Use of a Knowledge Base of Mathematical Theorems and Definitions

Benjamin Shults

April 1995

ATP 127
The Creation and Use of a Knowledge Base of Mathematical Theorems and Definitions*

Benjamin Shults
Department of Mathematics
University of Texas at Austin
Austin, TX 78712
bshults@math.utexas.edu

April 11, 1995

Abstract

IPR is an automatic theorem proving system intended particularly for use in higher-level mathematics. It discovers the proofs of theorems in mathematics applying known theorems and definitions. Theorems and definitions are stored in the knowledge base in the form of sequents rather than formulas or rewrite rules. Because there is more easily-accessible information in a sequent than there is in the formula it represents, a simple algorithm can be used to search the knowledge base for the most useful theorem or definition to be used in the theorem-proving process. This paper describes how the sequents in the knowledge base are formed from theorems stated by the user and how the knowledge base is used in the theorem-proving process. An example of a theorem proved and the English proof output are also given.

1 Introduction

The motivating goal behind this work is to develop a theorem-proving system which will be useful to both an expert and a non-expert in the attempt to prove theorems in advanced pure and applied mathematics. A goal of the project is to allow the user to proceed through a well-structured textbook having the system prove the theorems in succession, allowing the user to rely almost completely on the expertise of the author of the mathematics textbook or article. The degree of success of the work is illustrated by its ability to prove theorems from graduate level mathematics textbooks. See Section 6 for an example.

*This work has been supported in part by the National Science Foundation (grants CCR 9401873 and CCR 9307731).
Other automatic (or semi-automatic) theorem provers (e.g. [26, 27]) have proved some theorems in theories such as set theory or group theory by including in the hypotheses of the theorem enough information, such as the axioms, so that the theorem becomes a theorem of first- or higher-order logic. Although IPR can prove theorems of first-order logic, the approach we have generally taken with IPR is not to put any additional information into the hypotheses of a theorem. Theorems given to IPR are just as they would appear in a textbook in mathematics.\footnote{In fact, they are expressed in a language more formal than natural language (Section 2.1). Don Simon [30] has had success in translating from natural language to a more formal language which can be used by a prover.} It follows that they are not theorems in first-order logic and their proofs rely on the use of axioms, theorems and definitions—putatively satisfiable—from a knowledge base.

This is the way IPR proves theorems. Given a theorem to prove, IPR examines other theorems and definitions, which the user has loaded into the knowledge base, trusting that they are true in some model. The theorems proved by IPR in this way will only be known to be true in the models of the theorems used to prove them [12, 22].

One cannot get very far into mathematics without the notions of class and membership.\footnote{There are other primitive notions, such as mappings, on which the theories of mathematics could be based. Even so, the notions of class and membership quickly become useful and necessary.} IPR applies some very basic and flexible schemata which allow it to begin to reason about classes and members. These schemata are the mechanism by which IPR is able to prove theorems of type theory or “higher-order logic”. These are described in Section 2.2.

The combination of a sequent-calculus handling of first-order logic and equality, the use of these schemata and the automatic recall of known theorems and definitions enable IPR to express and attempt to prove any theorem in ordinary mathematics. With interaction, it should be able to prove any theorem in ordinary mathematics.

The handling of equality in the system is briefly described in Section 2.3.

Section 3 describes how theorems are reduced and stored in the knowledge base. Section 4 describes how theorems from the knowledge base are applied to sequents. Section 5 describes some algorithms for selecting theorems from the knowledge base. Section 6 contains a discussion of the Theorem N challenge and includes an example of a theorem proved completely automatically by IPR, using a very simple fetching algorithm, from a graduate-level textbook in topology.

\section{Knowledge Representation and Reasoning}

This section describes the way knowledge is represented in IPR and the way reasoning is performed. The notation $A^b_a$ represents the expression obtained by replacing every occurrence of the symbol $a$ in $A$ with the symbol $b$. 

...
2.1 The Language

Although IPR’s output is in English, the formulas which IPR reads are not completely in natural language. Formulas in IPR are standard first-order logic formulas in prefix notation with parentheses. This notation is sometimes called “S-expression” syntax. This language is extended by the classifier: (the-class-of-all (x) A) which is often written \{x : A\} in natural-language texts. The language also includes the \( \iota \)-operator (also called the “definite descriptor”): (the (x) A) which is usually written \( \iota_x A \) in natural language.

2.1.1 Syntax

This section describes the structure of formulas acceptable to IPR. A term and a formula in the language are defined recursively in terms of one another.

- If \( x \) is a variable, then \( x \) is a term.
- If \( f \) is an \( n \)-ary function symbol, then \( (f \ a_1 \ a_2 \cdots \ a_n) \) is a term if \( a_i \) is a term for each integer \( 1 \leq i \leq n \). If \( f \) is a function symbol of no arguments then \( (f) \) is called a constant term.
- If \( A \) is a formula and \( x \) is a variable then (the-class-of-all (x) A) is a term and (the (x) A) is a term.
- If \( P \) is an \( n \)-ary predicate symbol, then \( (P \ a_1 \ a_2 \cdots \ a_n) \) is a formula if \( a_i \) is a term for each \( 1 \leq i \leq n \).
- If \( A, B \) and \( C \) are formulas then so are \( (\text{not} \ A), (\text{iff} \ A \ B), (\text{if} \ A \ B \ C) \) and \( (\text{implies} \ A \ B) \).
- If \( A_1, A_2, \ldots, A_n \) are formulas then so are \( (\text{and} \ A_1 \ A_2 \cdots A_n) \) and \( (\text{or} \ A_1 \ A_2 \cdots A_n) \).
- If \( A \) is a formula and \( x \) is a variable, then \( (\forall x (x)) \ A \) and \( (\exists x (x)) \ A \) are formulas.
- If \( a \) and \( b \) are terms then \( (a = b) \) is a formula.

The symbols \( \text{true} \) and \( \text{false} \) are predicates of no arguments. The symbols \( \text{an-element} \) and \( \text{a-class} \) are predicates of one argument. The symbol \( \text{a-member-of} \) is a predicate of two arguments.

The symbols \( x \) and \( y \) are commonly-used variables although almost any symbol can be used as a variable. Let \( (\forall x (x_1) \cdots (x_n)) \ A \) abbreviate \( (\forall x_1 \ (\forall x_2 \ (\cdots \ (\forall x_n \ A) \cdots)) \) and similarly for \( (\exists x_1 \ (\exists x_2 \ (\cdots \ (\exists x_n \ A) \cdots)) \).
2.1.2 Semantics

Interpretations are built up in the standard way. The formula \( (\text{iff } A \ B) \) means that \( A \) and \( B \) are equivalent. The formula \( (\text{implies } A \ B) \) means that \( A \) materially implies \( B \). The formula \( (\text{if } A \ B \ C) \) means “if \( A \), then \( B \), otherwise \( C \)”. The formula \( (\text{and } A_1 A_2 \ldots A_n) \) is the conjunction of \( A_1, A_2, \ldots, A_n \). The symbol \( \forall \) is the universal quantifier and the symbol \( \exists \) is the existential quantifier. The symbol \( = \) is the identity predicate.

2.2 Sequent Calculus

The proof procedure used by IPR is the sequent calculus [14, 17, 19, 31].

A sequent is an ordered pair \( (\Gamma, \Delta) \) where \( \Gamma \) and \( \Delta \) are finite sets of formulas. We denote this sequent by \( \Gamma \rightarrow \Delta \). \( \Gamma \) is the set of hypotheses and \( \Delta \) is the set of conclusions or goals in the sequent.

If one of the sets of formulas is empty then it may be omitted and if one of the sets is a singleton then we simply write the formula. If \( X \) is a formula, then rather than writing \( \{X\} \cup \Gamma \rightarrow \Delta \) we will write \( X, \Gamma \rightarrow \Delta \) and similarly for formulas in the hypotheses.

Given a sequent \( \Gamma \rightarrow \Delta \), think of \( \Gamma \) as the conjunction of its elements and \( \Delta \) as the disjunction of its elements. Then the sequent as a whole is thought of as asserting that the conjunction of the hypotheses implies at least one of the conclusions.

The validity of a sequent \( S = \Gamma \rightarrow \Delta \) is shown by constructing a finite tree rooted at \( S \) using the rules presented in this paper. The sequent is shown to be valid if at some point every leaf of the tree is known to be valid. We use the following axiom schemata and inference rules in constructing the deduction tree.

**Axioms**

\[
X, \Gamma \rightarrow X, \Delta
\]

\[
(\text{falsity}), \Gamma \rightarrow \Delta
\]

\[
\Gamma \rightarrow (\text{truth}), \Delta.
\]

**\( \alpha \) - and \( \beta \) -rules**

**Negation Rules**

\[
\Gamma \rightarrow \Delta, X \\
\hline
\Gamma, (\neg X) \rightarrow \Delta
\]

\[
\Gamma, X \rightarrow \Delta \\
\hline
\Gamma \rightarrow (\neg X), \Delta
\]
Conjunction Rules

\[ \Gamma, X_1, \ldots, X_n \vdash \Delta \quad \Gamma \vdash \Delta, X_1 \ldots \Gamma \vdash \Delta, X_n \quad \Gamma \vdash \Delta, X \ldots \Gamma \vdash \Delta, X \]

Disjunction Rules

\[ \Gamma \vdash X_1, \ldots, X_n, \Delta \quad \Gamma \vdash (\text{or } X_1 \ldots X_n), \Delta \]

Implication Rules

\[ \Gamma, Y \vdash \Delta \quad \Gamma \vdash X, \Delta \quad \Gamma \vdash \Delta, Y \quad \Gamma \vdash \text{(implies } X \ Y) \vdash \Delta \]

Equivalence Rules

\[ \Gamma, Y, X \vdash \Delta \quad \Gamma \vdash Y, X, \Delta \quad \Gamma \vdash \Delta, Y \quad \Gamma \vdash Y, \Delta \]

If Rules

\[ \Gamma \vdash X, Z, \Delta \quad \Gamma \vdash X, Y, \Delta \quad \Gamma \vdash X, Y \quad \Gamma \vdash Z, \Delta, X \quad \Gamma \vdash \text{(if } X \ Y \ Z) \vdash \Delta \]

\( \gamma \)-rules

\[ \Gamma, A(x) \vdash \Delta \quad \Gamma \vdash \text{(forall } (x) \ A) \vdash \Delta \]

\( \delta \)-rules

\[ \Gamma \vdash A(f(x_1 \ldots x_n)), \Delta \quad \Gamma \vdash \text{(for-some } (x) \ A) \vdash \Delta \]

where \( x' \) is a new free variable.

The Tree Substitution Rule

If \( G \) is a proof tree then for any substitution \( \sigma \) of free variables in \( G \) we may apply the substitution across the entire tree simultaneously.

The Tree Substitution Rule is only applied when its use will close a branch or allow a theorem to be applied.

Soundness and completeness of these rules for first-order logic is proved elsewhere [17, 19, 31]. The particular \( \delta \)-rule used was proved correct by Hähnle [19]. IPR also uses a mixture of rigid and universal variables [4].
In order to keep from running forever, a limit is set on the number of times the $\gamma$-rule can be applied to any formula. The user may reset this limit at any time. This limit is referred to as the Q-limit.\footnote{The knowledge base of theorems and definitions will be searched when all other rules have been applied, the Q-limit has been reached and there is still a goal which is not known to be true. See Section 4.}

The sequent-calculus proof procedure is complete for first-order logic. In order to handle mathematics we need a higher-order language. In IPR, this is handled by the use of the comprehension schema. This can be stated as follows:

\textbf{SCHEMA 2.1} If $x$ is a variable and $A$ is a formula, then $\{x : A\}$ is a class. Further, if $a$ is an ar-element containing no free variables which are bound in $A$ and $a$ is a set or an ar-element then $a \in \{x : A\}$ if and only if $A_a^x$.

This schema is applied in the IPR system by the use of the following schemata.

\textbf{SCHEMA 2.2} (Comprehension in the Hypotheses)

\[\frac{\text{(an-element } x), A^y_x, \Gamma \rightarrow \Delta}{\text{(a-member-of } x\text{ (the-class-of-all } (y) A)), \Gamma \rightarrow \Delta}\]

where $x$ is a term, $y$ is a variable and $A$ is a formula.

\textbf{SCHEMA 2.3} (Comprehension in the Conclusions)

\[\frac{\Gamma \rightarrow \text{(an-element } x), \Delta}{\Gamma \rightarrow \text{(a-member-of } x\text{ (the-class-of-all } (y) A)), \Delta}\]

where $x$ is a term, $y$ is a variable and $A$ is a formula.

\textbf{SCHEMA 2.4} (Class)

Any sequent of the form $\Gamma \rightarrow \text{(a-class (the-class-of-all } (y) A)), \Delta$ where $y$ is a variable and $A$ is a formula is valid.

And finally, a schema is needed for handling the definite descriptor.

\textbf{SCHEMA 2.5} (io\textit{ta} removal) Any formula $(P \ t_1 \cdots t_n)$ which contains a subterm $s = (\text{the } (x) A)$ can be replaced by the formula

\[\begin{align*}
&\text{(if (for-some ((r)) (and $A^y_x$ (an-element $r$) } \\
&\text{(forall ((n)) (implies (and $A^y_n$ (an-element $n$) } \\
&\text{$(\equiv r\ n)$))))))} \\
&\text{(forall ((c)) (implies (and $A^y_c$ (an-element $c$) } \\
&\text{$(P \ t_1 \cdots t_n)^c$))}
\end{align*}\]
where $U$ is some constant (the default value for $U$ is the-empty-set) although its value is not significant.

Notice that these schema do not bind the user to a particular choice of axioms of set theory. The user may chose to select ZF or NGB, to allow ur-elements or not, or to be unspecific about the low-level axioms.

2.3 Equality

This section contains a discussion of the methods of handling equality in IPR.

The method used is complete for ground equality. That is to say that if a combination of unification and ground equality reasoning using the equalities in the hypotheses could prove the sequent to be valid then IPR will succeed with no further expansion of the tree. This is accomplished by the congruence closure technique [21, 24]. The congruence closure method needs to be extended to handle class- and iota-expressions. That is to say that to check for equality between such terms, the formulas in the terms need merely to 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.

There are times when actual substitution of equals needs to occur when reasoning in the presence of theorems and class-expressions. Therefore, a very restricted form of Fitting’s rules for equality [14] (similar to paramodulation [28]) should be used. For example, in the formula (a-member-of $s \ X$) the equality reasoner tries to rewrite $X$ as a class expression if possible because this allows further reasoning.

Because of the presence of rigid variables in the sequent calculus many of the well-known methods for handling equality are not applicable [4]. For example, there are ways of extending the congruence closure method to make it complete for a larger class of sequents (e.g. McAllester’s fast grammar rewriting [21]) but most of these work with strictly universal variables.

Beckert [4], Fitting [14] and Gallier [16] have developed complete methods for handling equality in the presence of rigid variables. Some of these methods may become a part of IPR in the future.

3 Representation of Knowledge in the Knowledge Base

This section explains how IPR forms and stores knowledge about mathematical concepts. Theorems and definitions are stored in the knowledge base as sequents. This device raises the content of an “atomic” datum above the level of literal or formula. This enables a relatively simple fetching algorithm to have success in selecting the ideal theorem or definition for use in a proof. However,
the rules applied to formulas being installed into the knowledge base are slightly different from the ordinary rules.

The formula $X$ to be installed into the knowledge base is automatically broken down by IPR into smaller bits of knowledge by the following procedure.

First, IPR forms the sequent $\rightarrow X$ and applies all $\alpha$- and $\beta$-rules possible.

The following rules are also applied only in this situation.

### $\delta^{-1}$-rules

\[
\frac{\Gamma, A'_x, (\forall (x) A) \rightarrow \Delta}{\Gamma, (\forall (x) A) \rightarrow \Delta} \quad \frac{\Gamma \rightarrow \Delta, A'_x, (\exists (x) A)}{\Gamma \rightarrow \Delta, (\exists (x) A) A}
\]

where $A'$ is the result of applying a first-order logic equivalence in order to push the quantifier into the formula $A$ if possible. If the quantifier cannot be pushed in then this rule cannot be applied.

### $\gamma^{-1}$-rules

\[
\frac{\Gamma, A''_x \rightarrow \Delta}{\Gamma, (\exists (x) A) \rightarrow \Delta} \quad \frac{\Gamma \rightarrow \Delta, A''_x}{\Gamma \rightarrow \Delta, (\forall (x) A) A}
\]

where $x'$ is a new free variable.

Notice that the $\gamma^{-1}$-rule applies at times when the ordinary $\delta$-rule applies and that the $\delta^{-1}$-rule applies when the ordinary $\gamma$-rule applies.

**EXAMPLE 3.1** Suppose a user wants the following axiom to be in the knowledge base.

(def-axiom extensionality
  (iff (forall ((u))
    (iff (a-member-of u x)
      (a-member-of u y)))
    (= x y))
  (string "the axiom of extensionality"))

IPR creates the following sequent: $\rightarrow X$ where

$X = (\iff (forall ((u)) (iff (a-member-of u x) (a-member-of u y)))
  (= x y))$

and transforms it into the following sequents using $\alpha$- and $\beta$-rules:

$S_1 = X_1 \rightarrow (= x y)$ where

$X_1 = (forall ((u)) (iff (a-member-of u x) (a-member-of u y)))$

and

$S_2 = (= x y) \rightarrow X_1$. 

8
Then it applies the $\delta^{-1}$-rule and an $\alpha$-rule to $S_1$ to obtain

$$S_3 = X_2, X_3 \rightarrow (= x y)$$

where

$$X_2 = (\forall (u) \,(\Rightarrow (a\text{-member-of } u x) \,(a\text{-member-of } u y)))$$

and

$$X_3 = (\forall (u) \,(\Rightarrow (a\text{-member-of } u y) \,(a\text{-member-of } u x))).$$

At this point, no more rules may be applied to $S_3$ so it is added to the knowledge base.

IPR then applies a $\gamma^{-1}$-rule to $S_2$ to obtain

$$ (= x y) \rightarrow (iff \,(a\text{-member-of } u1 x) \,(a\text{-member-of } u1 y)).$$

Then it applies $\alpha$- and $\beta$-rules to obtain

$$S_4 = (= x y), (a\text{-member-of } u1 x) \rightarrow (a\text{-member-of } u1 y)$$

and

$$S_5 = (= x y), (a\text{-member-of } u1 y) \rightarrow (a\text{-member-of } u1 x).$$

At this point, no more rules can be applied to the sequents $S_4$ and $S_5$ so they are added to the knowledge base.

### 3.1 Using Predicate Definitions

When the user enters a predicate definition into the knowledge base such as:

```
(def-predicate (a-locally-compact-top-space x_)
  (forall ((x))
    (implies (a-member-of x x_)
      (for-some ((c))
        (and (a-nbhd-of c x x_)
          (a-compact-subset-of c x_))))
    ))
```

the system currently treats it just as if it were the theorem:

```
(iff (a-locally-compact-top-space x_)
  (forall ((x))
    (implies (a-member-of x x_)
      (for-some ((c))
        (and (a-nbhd-of c x x_)
          (a-compact-subset-of c x_)))))
  )
```

At present, the user is expected to keep the knowledge base as consistent as he sees fit. If desired, a sophisticated syntax-checker could be installed.

The string argument in the definition is used for English output.
3.2 Using Term Definitions

When the user enters a term definition into the knowledge base such as:

\[
\begin{align*}
\text{(def-term (the-product-over } x_\_ a_\_) \\
\text{(the-class-of-all } x \text{) }
\end{align*}
\]

\[
\begin{align*}
\text{(and (a-function-on } x a_\_) \\
\text{(forall } ((a)) \\
\text{ (implies (a-member-of } a a_\_) \\
\text{ (a-member-of (apply x a) (apply x_ a)))))}
\end{align*}
\]

(string "the Cartesian product of " a over the index set "a")

the system adds the theorem

\[
\begin{align*}
\text{(=} (the-product-over } x_\_ a_\_) \\
\text{(the-class-of-all } x \text{) }
\end{align*}
\]

\[
\begin{align*}
\text{(and (a-function-on } x a_\_) \\
\text{(forall } ((a)) \\
\text{ (implies (a-member-of } a a_\_) \\
\text{ (a-member-of (apply x a) (apply x_ a)))))}
\end{align*}
\]

to the knowledge base. The prover treats equality theorems specially. When the terms involved in an equality theorem are present in a sequent, then the equality can be added to the congruence grammar of that sequent. The equality can also be used to rewrite terms into class expressions when it is likely to help the reasoning process.

4 Using Theorems from the Knowledge Base

This section describes how theorems are applied to sequents which are being proved by IPR. The knowledge base will be searched when all other rules have been applied, the $Q$-limit has been reached and there is still a sequent $S = \Gamma \rightarrow \Delta$ which is not known to be true.

Section 5 describes how a theorem may be selected from the knowledge base. For now, suppose that the sequent $T = U \rightarrow V$ has been selected from the knowledge base for use in establishing the truth of the sequent $S$. The application of the following rule preserves truth, that is to say that if a sequent is proved by these rules using a set $A$ of theorems from the knowledge base, then we know that the sequent is true in any model of $A$.

First we give the most general rule. This rule can soundly be applied using any theorem from the knowledge base.

**Theorem Application Rule (most general)**

\[
\frac{(or V_1 \cdots V_n), \Gamma \rightarrow \Delta, \Gamma \rightarrow \Delta, (and U_1 \cdots U_m)}{\Gamma \rightarrow \Delta}
\]
where \( V = \{V_1, \ldots, V_n\} \) and \( U = \{U_1, \ldots, U_n\} \) and \( U \to V \) is in the knowledge base.

The application of the above rule in a systematic manner provides a complete method for using all theorems in the knowledge base. In effect, this rule simply adds the theorem to the sequent. Therefore, the systematic application of this rule to each theorem in the knowledge base would be basically equivalent to adding the entire knowledge base to the hypotheses of the theorem being proved. However, the approach taken in IPR is different. IPR restricts the Theorem Application Rules so that a theorem from the knowledge base is used only when it has something in common with the sequent being proved. Therefore, the most general rule is never applied by IPR without some restriction.

Here are the restrictions on Theorem Application Rules 1–4.

- Rules 1–4 only apply to a theorem \( U \to V \) in the knowledge base in case there are sets \( U', V', \Delta' \) and \( \Gamma' \), some of which must be nonempty, such that \( U' \subseteq U \), \( \Gamma' \subseteq \Gamma \), \( V' \subseteq V \) and \( \Delta' \subseteq \Delta \). Further, there must be a substitution \( \sigma \) such that \( U' \sigma = \Gamma' \sigma \) and \( V' \sigma = \Delta' \sigma \).

- Let \( M = V \setminus V' \) and \( N = U \setminus U' \) so that \( M \) is the set of formulas in \( V \) which were not successfully unified with a formula in \( \Delta \) and similarly for \( N \). Rules 1–4 will only apply to a theorem if \( M \cap \Gamma = \emptyset \) and \( N \cap \Delta = \emptyset \). If either of these intersections is nonempty then the use of the theorem does not help, but rather adds useless branching to the deduction tree.

- The substitution \( \sigma \) which unifies the sets \( U \) and \( \Gamma \) and the sets \( V \) and \( \Delta \) must be applied using the Tree Substitution Rule.

When the above conditions are satisfied, the following rules may be applied.

**Theorem Application Rule 1**

If \( M \) and \( N \) are nonempty then the following rule may be applied

\[
\frac{(\text{or } M_1 \cdots M_m), \Gamma \to \Delta, \Gamma \to \Delta', \text{(and } N_1 \cdots N_n)}{\Gamma \to \Delta}
\]

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

**Theorem Application Rule 2**

If \( N \) is empty and \( M \) is not then the following rule may be applied

\[
\frac{(\text{or } M_1 \cdots M_m), \Gamma \to \Delta}{\Gamma \to \Delta}
\]

where \( M = \{M_1, \ldots, M_m\} \).
Theorem Application Rule 3
If $M$ is empty and $N$ is not then the following rule may be applied

$$
\frac{\Gamma \rightarrow \Delta, (\text{and } N_1 \ldots N_n)}{\Gamma \rightarrow \Delta}
$$

where $N = \{N_1, \ldots, N_n\}$.

Theorem Application Rule 4
If $M = \emptyset$ and $N = \emptyset$ then the following rule may be applied

$$
\Gamma \rightarrow \Delta
$$

Notice that in rules 1–4 the amount of eventual branching after the application of the rule will be $n + m$.

Informally speaking, the more formulas from the theorem which are successfully unified with the sequent, the larger a step the theorem application rule is taking (in the same sense that hyper-resolution is a larger step, the more literals involved). Also, the fewer unsuccessfully-unified formulas from the theorem, the less branching will occur. In the limiting case, when all formulas from the theorem are unified with formulas from the sequent, the sequent is proved immediately. These ideas will be used by the fetching algorithms in Section 5.

The Theorem Application Rules 1–4 are not complete. That is to say that it is possible that the sequent is a consequence of the theorems in the knowledge base but these rules will not be able to establish that. It has been mentioned that the most general Theorem Application Rule is complete. There are ways of restricting the application of the most general rule so that the theorems used are more likely to be relevant to the problem at hand and yet retain completeness. For example, the most general Theorem Application Rule need only be applied when the theorem contains a predicate symbol identical to some predicate symbol in the sequent and they are located in positions of the same polarity. More restrictive conditions would be sufficient for completeness but we are not concerned with this issue here since IPR does not systematically use each applicable theorem but rather tries to choose intelligently.

That Rules 1–4 are equivalent to the most general rule in the mentioned circumstances, is easy to show. Therefore, it will be sufficient to show that the most general rule preserves truth.

**THEOREM 4.1 (Soundness of Theorem Use)** Suppose a theorem is entered into the knowledge base and produces in the knowledge base the sequent $U \rightarrow V$. Then $\Gamma \rightarrow \Delta$ is true in every model of $U \rightarrow V$ if and only if $(\text{or } V_1 \cdots V_n), \Gamma \rightarrow \Delta$ and $\Gamma \rightarrow \Delta, (\text{and } U_1 \cdots U_n)$ are true.

Proof idea: Notice that the following formula in the language of IPR is a tautology.
\[
(\text{implies} (\text{implies} (\text{and} (U1) (U2)) (\text{or} (V1) (V2)))
(\text{iff} (\text{and} (\text{implies} (\text{and} (\text{or} (V1) (V2)) (\text{Gamma})) (\text{Delta}))
(\text{implies} (\text{Gamma}) (\text{or} (\text{and} (U1) (U2)) (\text{Delta})))))
(\text{implies} (\text{Gamma}) (\text{Delta})))
\]

It must also be shown that the interaction between the rules for creation of sequents in the knowledge base and the rules for unifying them with the sequent in question are sound.

5 Fetching Algorithms

The implementer of this method is very free to experiment with fetching algorithms. Section 5.1 describes a simple algorithm, Simp, whose success is illustrated in Section 6. Section 5.2 gives some recommendations for improvements to Simp and reports the results of some of these improvements. It is to be expected that better algorithms will give more frequent and faster success even though they are expensive. Section 5.3 briefly describes the more sophisticated Gazing [2] algorithm.

5.1 A Simple Fetching Algorithm

The simple fetching method presented in this section is successful in domains where other methods—such as rewriting—are not useful. However, this simple method alone is least useful in domains in which rewriting is very successful. The reasons for this will be mentioned later. That such a simple fetching algorithm can be successful at all (see Section 6) shows the benefit of the method of storing and using knowledge. We will call this simple algorithm Simp.

As was mentioned before, the knowledge base will be searched when all other sequent rules have been applied, the Q-limit has been reached and there remains a sequent \( \Gamma \rightarrow \Delta \) which is not known to be true. Then for each sequent \( U \rightarrow V \) in the knowledge base, unify as many as possible of the elements of \( U \) with the elements of \( \Gamma \), then unify as many as possible of the elements of \( V \) with the elements of \( \Delta \). The substitutions which unify these must be consistent, that is, for each sequent in the knowledge base there must be a single substitution which accomplishes all of these unifications. Once this is done, then for each sequent \( T \) in the knowledge base, the following information will be available: a substitution which unifies part (or all) of it with the sequent \( \Gamma \rightarrow \Delta \), the number of goals and hypotheses in \( T \) which were successfully unified and the number of goals and hypotheses in \( T \) which were not unified with any formula in \( \Gamma \rightarrow \Delta \).\footnote{The sequent \( T \) can have more than one way to unify with \( \Gamma \rightarrow \Delta \). I recommend that the unifier backtrack in order to find every possible way each theorem applies. This may be expensive but without it, the procedure can miss very easy proofs. The results in Section 6 were obtained without backtracking.}
Using this information, each sequent in the knowledge base is given a rating according to the following guidelines:

- If all goals and hypotheses of the sequent $T$ were unified with some formula in $\Gamma \rightarrow \Delta$ then $T$ gets the highest possible ranking (because it immediately shows that $\Gamma \rightarrow \Delta$ is true using Theorem Application Rule 4). If there are more than one or no sequents in the knowledge base with this property then the sequents are judged further with the following guidelines.

- The fewer of the variables in $\Gamma \rightarrow \Delta$ which are eliminated by the substitution the better. This is so because, according to the Tree Substitution Rule, any substitution must be applied across every sequent in the tree.

- The fewer unsuccessfully-unified formulas in $T$ the better.

- The more successfully-unified formulas from $T$ the better.

Recall (Section 4) that the eventual amount of branching depends on the total number of unsuccessfully-unified formulas from the knowledge-base theorem. The third guideline encourages less branching.

The fourth guideline encourages big steps to be taken. Put another way, it encourages the use of theorems which are dealing closely with the topic of the sequent.

The choices made in Simp were intended to make it generally applicable rather than especially suited to a particular domain. Section 6 gives an example of the type of theorem on which Simp is successful.

5.2 Improvements to the Simple Algorithm

The algorithm presented in the previous section, although somewhat successful, has difficulties. Many of these can be overcome by the addition of more sophisticated methods.

For example, when a theorem was loaded into the data-base, no information was provided by the user concerning how or when it should be used. The addition of such information may help IPR as it does other systems such as Boyer and Moore’s NQTHM [10].

Furthermore, each time a theorem was needed, the prover examined every piece of knowledge in the knowledge base before selecting what it thought to be the best. There are many ways of narrowing the search space. One way is to partition the knowledge base in some way. A more generally-applicable and yet inexpensive way is to allow IPR to search the entire knowledge base the first time and make a list of all theorems which contain formulas which unify with any formula in the sequent. The next time the knowledge base needs to be searched, only those theorems will be examined. If new formulas have been added to the goal sequent, then the rest of the knowledge base only needs to be
searched for theorems which unify with the new formulas. This improves IPR’s performance tremendously in large knowledge bases.

There are other guidelines which could be added to the list in Section 5.1. These guidelines vary in the cost of application.

- Prefer not to use a theorem which introduces a new predicate.
- Prefer not to use a theorem which introduces a new function symbol.
- Prefer to use a theorem which adds equality formulas to the hypotheses.
- Prefer to use a theorem which introduces a formula of the form \((a\text{-member-of } t \ (\text{the-class-of-all } (x) A))\).
- Keep track of the theorems applied and the details of the application and do not allow a theorem to be applied in the same way again until other things have been tried.
- If the formula \(F \in \Gamma \rightarrow \Delta\) is successfully unified with a theorem from the knowledge base and that theorem is applied, then prefer not to use theorems which successfully unify with \(F\) in the future.

The last guideline encourages the prover to use the new knowledge acquired from theorem use instead of applying several different theorems about one topic. In the language of rewriting we would say that \(F\) is a formula which has been rewritten in a hopefully simpler way and therefore we should concentrate on the result of the rewriting rather than rewriting it again in some other way.

Another method for choosing theorems from a knowledge base is the Peeking technique [8]. The Peeking technique chooses a theorem by seeing if the result may help in the proof. This is done in a relatively unsophisticated way and is not as discriminating as Simp. In particular, the Peeker only looks at the top level predicates of the theorems.

Some proof-checking systems, such as IMPS [13] and HOL [18], use known theorems and definitions in a more user-directed manner. IPR can operate in a very informative interaction mode in which the user can select theorems to apply.

5.3 The Gazing Algorithm

The Gazing algorithm was developed by David Barker-Plummer [2, 3]. The Gazing technique fills in a weakness of Simp and other methods of fetching theorems. It accomplishes this by making a complete plan (which may or may not work) for the proof. Its plan consists of a list of theorems to apply in order along with some instructions about how to apply them. The Gazing algorithm contains ideas which will probably be very important to the future of automatic theorem proving.
One of the advantages—lacking in Simp—obtained by Gazing is the tight control it gives over the application of rewrite rules. A well-known way of applying theorems involves using rewrite rules to rewrite terms and predicates toward a normal form. In the rewrite-rule technique, the theorem (usually involving equality or a relation with related properties) is translated into a rewrite rule or a conditional rewrite rule. As long as the rules satisfy certain properties, they can safely be used at any time to simplify the sequent being attacked. Simp does not guide the application of rewrite rules, which occur as theorems in the knowledge base. It simply applies the theorem every time it decides that it is the best theorem from the knowledge base. Without guidance, Simp can waste a tremendous amount of time rewriting in both directions without simplifying the problem if the rewrite rules in the knowledge base seem more applicable to it than other theorems. Many of the rewrite-rule methods could be combined with the knowledge base and sequent based rules presented in this paper.

Simp works well in many situations where rewrite rules do not naturally apply. Because theorems are stored in the form of sequents, all of the information in the theorem is naturally available at a high level of accessibility. Therefore, Simp may have an advantage over Barker-Plummer’s implementation in domains in which 1) theorems do not involve equality or other predicates which prove the worth of rewrite rules, 2) the statements of theorems are long and involve many formulas, 3) the knowledge base is large or 4) there is a large number of predicates in the theory.

Barker-Plummer’s particular application of the proof-planning ideas limited their use to domains which satisfied the four conditions above. His implementation of the ideas also gave completeness a rather low priority. There are other differences in the way IPR and Gazer store knowledge, reason in first-order logic and apply theorems. However, a significant advance which Gazer made was in its automatic construction and use of plans: the Gazing technique.

The incorporation of the ideas behind Gazing into the more generally-applicable framework of IPR would profoundly affect IPR’s success in domains where it is currently weak. This appears to be a very promising direction. The addition of Gazing to IPR would control IPR’s use of rewrite rules and, in other domains, give IPR more far-sighted guidelines for choosing the best theorem to apply. Since Gazing appears to be able to fill in what is lacking in the near-sighted algorithms like Simp, this combination ought to allow a prover like IPR to accomplish the Theorem N Challenge (Section 6) for more values of N.

However, even with Gazing, there would still remain a large class of difficult theorems which IPR could not handle automatically. These include theorems which require the prover to make a (sometimes very clever) guess for a set-variable’s instantiation. Proving these would require better higher-order logic reasoning capabilities such as extensions of the ideas in provers such as Feng and Bledsoe’s Set-Var [9] or the use of a higher-order method such as that of Peter Andrews’ TPS [1].
6 The Theorem $N$ Challenge

The Theorem $N$ Challenge has been around for decades motivating much of the work of Woodrow Bledsoe, Don Loveland and others who have been applying automated theorem proving to mathematics.\(^5\) Here is one way of wording the challenge:

**CHALLENGE 6.1** Have the program prove the $N^{th}$ theorem in a mathematics text using previous theorems and definitions in the text.

It seems reasonable in practice for the user of the prover to have the theorems and definitions of the text cataloged by section. Thus, when the user wants to prove a theorem about the product of locally-compact topological spaces, he could easily load information from the sections of the book covering compact spaces, product spaces, locally-compact spaces and some basic set theory; that is, all significant statements from all sections which the user thinks may contain useful information previous to the $N^{th}$ theorem. Simp has shown that it can handle this problem well in many cases.

6.1 Example from Kelley

The results in this section were accomplished by IPR using only the Simp fetching algorithm (see Section 5.1).

The first example I wanted to attack with IPR was the 101$^{st}$ labeled statement in *General Topology* by John Kelley [20].\(^6\) This is Theorem 19 in Chapter 5. Here is the statement:

*If a product is locally compact, then each coordinate space is locally compact and all except a finite number of coordinate spaces are compact.*

Since the theorem was really a conjunction of two theorems, I will focus on the first.

Here is the statement in IPR syntax:

\[(\text{def-theorem kelley-5-19a})\]
\[\text{(implies (a-locally-compact-top-space)}\]
\[\text{(the-product-top-space-over X_ A_))}}\]
\[\text{(forall ((a))}\]
\[\text{(implies (a-member-of a A_)}\]
\[\text{(a-locally-compact-top-space (apply X_ a))))}}\]
\[\text{(string "the first half of Theorem 19 in Chapter 5 of Kelley")})\]

\(^5\)This was communicated to me by Woodrow Bledsoe.

\(^6\)The choice of $N=101$ and text were made due to a comment made by Robert Boyer in conversation.
The following three theorems are the only ones needed in the proof:

(def-theorem kelley-p-147e
  (implies (and (an-open-function-from-onto f a b)
               (a-continuous-function-from-onto f a b)
               (a-locally-compact-top-space a))
           (a-locally-compact-top-space b))
  (string "a statement near the bottom of page 147 of Kelley")
)

(def-theorem kelley-p-90a
  (a-continuous-function-from-onto
   (the-projection-function a x_ a_)
   (the-product-top-space-over x_ a_)
   (apply x_ a))
  (string "the statement on the top of page 90 of Kelley")
)

(def-theorem kelley-3-2
  (an-open-function-from-onto
   (the-projection-function a x_ a_)
   (the-product-top-space-over x_ a_)
   (apply x_ a))
  (string "Theorem 2 in chapter 3 of Kelley")
)

The string argument is used for English output.

If these are the only theorems in the knowledge base then the theorem is proved easily by Simp. A seven-step proof was found in 0.14 seconds.

Here is the output of the program describing the steps taken in the proof found by IPR with Simp. The notation .f x refers to the application of the function f to the element x.

Suppose that

the product topological space of _X_ over the index set
_A_ is a locally-compact topological space.

Show that

for every A
if
   A is a member of _A_
then
   _X_
A is a locally-compact topological space.

Let _A be a constant and show that
if
\( A \) is a member of \\
\( A \)
then
\( X \)
\( A \) is a locally-compact topological space.

Suppose that \\
\( A \) is a member of \\
\( A \)
Show that \\
\( X \)
\( A \) is a locally-compact topological space.

Since we know that \\
the product topological space of \\
\( X \) over the index set \\
\( A \) is a locally-compact topological space \\
and we are trying to show that \\
\( X \)
\( A \) is a locally-compact topological space \\
we can apply a statement near the bottom of page 147 of Kelley.

Now we must show that \\
F is an open function from \\
the product topological space of \\
\( X \) over the index set \\
\( A \) onto \\
\( X \)
\( A \) and \\
F is a continuous function from \\
the product topological space of \\
\( X \) over the index set \\
\( A \) to \\
\( X \)
\( A \)

Split the goal \\
F is an open function from \\
the product topological space of \\
\( X \) over the index set \\
\( A \) onto \\
\( X \)
A and
F is a continuous function from
the product topological space of
\(X\) over the index set
\(A\) to
\(X\)
\(A\)
into cases.

After that split we will do the following:

1
Since we are trying to show that
\(F\) is an open function from
the product topological space of
\(X\) over the index set
\(A\) onto
\(X\)
\(A\)
we can apply Theorem 2 in chapter 3 of Kelley
which finishes the proof of that goal.

2
Since we are trying to show that
\(F\) is a continuous function from
the product topological space of
\(X\) over the index set
\(A\) to
\(X\)
\(A\)
we can apply the statement on the top of page 90 of Kelley
which finishes the proof of that goal.

The information in this output tells how the theorem was broken down and
then which theorems were applied [7].

Suppose the user had previously worked through the book carefully, proving
or letting IPR prove each theorem as he came to it. The user would save all of
the theorems in files organized by section. On the day the user gets to Theorem
5.19 of Kelley’s *General Topology* he will have thousands of theorems at his
disposal. He would notice that the current theorem deals with product spaces,
locally-compact spaces and the concept of infinity. Therefore, he would load
in the files containing the theorems from the sections of the book dealing with
these topics.

Now let us make the problem more realistic. It is clear that the theorem has
to do with Cartesian products, product spaces and locally-compact topological
spaces. Therefore, I loaded the theorems and definitions (labeled and otherwise) from the three sections of the text related to product spaces, locally-compact spaces and compact spaces, and asked the prover to prove the theorem. This seems to be a reasonable thing to expect an user of a theorem-proving program to be able to do. This totaled 27 theorems and 7 term definitions. I count predicate definitions as theorems. The same proof was found completely automatically in 6.45 seconds. Later, I put 82 theorems and 10 term definitions into the knowledge base from other related sections of the text and the same proof was found in 35.76 seconds, again, with no human interaction.7

Recall that these results were obtained with the most naive search techniques, i.e. Simp.

6.2 Other Theorems Proved

IPR has been used to prove theorems from several well-structured graduate-level and upper undergraduate-level textbooks in mathematics including Royden’s *Real Analysis* [29], Kelley’s *General Topology* [20], Spivak’s *Calculus on Manifolds* [32], Munkres’ *Topology* [23], Flanders’ *Differential Forms with Applications to the Physical Sciences* [15], Bishop and Goldberg’s *Tensor Analysis on Manifolds* [6] and Bernays’ *Axiomatic Set Theory* [5].

More difficult theorems can be proved by IPR using more sophisticated fetching algorithms. The most difficult theorems proved by IPR required the user to interact (by telling IPR which theorem to apply) or to put into the knowledge base certain key lines from the proof given by the author of the textbook. This latter style of human-computer “interaction” requires a bit less work from the user than what was described by Bledsoe in his article on interactive proof presentation [7] since this style requires the computer to decide when and how to apply the information present in the textbook proof.

7 Conclusion

The result of the experiment with the problem from Kelley’s *General Topology* is an example in which the prover shows “superhuman” performance. That is, even in the presence of a lot of knowledge, a very simple algorithm was able to find the shortest possible proof in about 36 seconds. The success which Simp has had is easily superseded by IPR using more complex fetching rules (Section 5.2).

I hope that the reader will find the ideas in this paper useful. The method is very flexible and can be combined with many other techniques. I hope this

---

7When IPR finishes a proof it examines the proof tree and automatically eliminates all branches which were not necessary. Therefore, some of the time taken in the latter two runs was spent 1) creating unneeded branches in the tree and 2) collapsing the proof when it was discovered which branches were unneeded (as in the HARP prover [25]). Without this feature, IPR is faster but the proof it presents to the user is considerably longer and less sensible.
will encourage others to build and experiment with provers which try to use
a knowledge base of theorems and definitions. This kind of technology will
allow a non-expert user to prove theorems in any well-structured textbook in
mathematics. In addition, these proofs are short and easy for a human to follow.

Acknowledgments

IPR’s knowledge base is managed by Algernon [11]. I would like to thank Ben
Kuipers and Micheal Hewett for their help with Algernon. I would like to thank
Woodrow Bledsoe and Larry Hines for their support and advice during every
stage of this work. I am also indebted to David Barker-Plummer (Gazing),
Reiner Hähnle (semantic tableaux) and Bernhard Beckert (universal variables
in semantic tableaux) for their answers to my questions about their work.

References

[1] Peter B. Andrews, Matthew Bishop, Sunil Issar, Dan Nesmith, Frank Pfen-
ning, and Hongwei Xi. TPS: A theorem proving system for classical type
type. Technical Report 94–166, Department of Mathematics, Carnegie
Mellon University, 1994.


D. Kapur, editor, *Proceedings of the Eleventh International Conference on
Automated Deduction*, volume 607 of *Lecture Notes in Computer Science*,

[4] Bernhard Beckert. A completion-based method for mixed universal and
Automated Deduction CADE, Nancy/France*, LNAI 814, pages 678–692.
Springer Verlag, 1994.

Dover edition, first published in 1991, is an unabridged and unaltered re-
publication 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*.

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.


