TPS: A THEOREM PROVING SYSTEM FOR CLASSICAL TYPE THEORY

by

Peter B. Andrews,

Matthew Bishop,

Sunil Issar,

Dan Nesmith,

Frank Pfenning,

and

Hongwei Xi

Research Report No. 94–166 ,
June, 1994 α

TPS: A Theorem Proving System for Classical Type Theory

Peter B. Andrews[1], *Matthew Bishop*[1], *Sunil Issar*[2], *Dan Nesmith*[3], *Frank Pfenning*[2], *Hongwei Xi*[1]

Abstract

This is a description of TPS, a theorem proving system for classical type theory (Church's typed λ-calculus). TPS has been designed to be a general research tool for manipulating wffs of first- and higher-order logic, and searching for proofs of such wffs interactively or automatically, or in a combination of these modes. An important feature of TPS is the ability to translate between expansion proofs and natural deduction proofs. Examples of theorems which TPS can prove completely automatically are given to illustrate certain aspects of TPS's behavior and problems of theorem proving in higher-order logic.

KEY WORDS: higher-order logic, type theory, mating, connection, expansion proof, natural deduction.

CONTENTS

1. Introduction

TPS is a theorem proving system for classical type theory (Church's typed λ-calculus [19]) which has been under development at Carnegie Mellon University for a number of years.[4] This paper gives a general description of TPS, serves as a report on our implementations of ideas which

[1]Mathematics Department, Carnegie Mellon University, Pittsburgh, PA 15213, U.S.A.

[2]School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, U.S.A.

[3]University of the Saarland, 66041 Saarbruecken, Germany

[4]This material is based upon work supported by the National Science Foundation under grant CCR-9201893 and previous grants.

were discussed in previous papers, and illuminates what can be accomplished using these ideas. Many of the ideas underlying **TPS** are summarized in [8], with which we shall assume familiarity.

We start with a brief history of the **TPS** project.

TPS is based on an approach to automated theorem proving called the *mating method* [5], which is essentially the same as the *connection method* developed independently by Bibel [13]. The mating method arose from reflections [3] on what a proof by resolution [48] reveals about the logical structure of the theorem being proved, but a distinguishing characteristic of the mating method is that it does not require reduction to clausal form.

Matings provide significant insight into the logical structure of theorems, but it is not always easy for people to grasp them intuitively or to relate them to other approaches to theorem proving, so a procedure for automatically transforming acceptable matings into proofs in natural deduction style was developed [4]. The ideas in [5], [4], and [30] were implemented in a theorem proving system which we now call **TPS1**. This was described in [35] and [6]. It automatically proved certain theorems of type theory (higher-order logic) as well as first-order logic, and embodied a proof procedure which was in principle complete for first-order logic, though not for type theory.

As a start toward extending of the mating method to a complete method for proving theorems of higher-order logic, it was shown in [6] that a sentence is a theorem of elementary type theory (the system of [1] and [2]) if and only if it has a tautologous *development*, where a development is the analogue of a Herbrand expansion of a sentence of first-order logic. Once one has found a tautologous development for a theorem, one can construct a proof of it in natural deduction style without further search. Thus, the problem of finding proofs for theorems of higher-order logic can be reduced to the problem of finding tautologous developments for them, and the search can be carried on in a context where one can hope to analyze the essential logical structure of the theorem.

Dale Miller explored this subject more deeply, proved an analogue of the metatheorem mentioned above in which the notion of a development was replaced by that of an *expansion proof*, gave the details of an explicit algorithm for converting expansion proofs into natural deduction proofs, and proved it correct. An expansion proof is an elegant and concise representation of a theorem of type theory, its tautologous development, and the relation between them. Matings are naturally embedded in expansion proofs, and Miller's work [36], [37], [38] provided a firm theoretical foundation for the extension of the matings approach to theorem proving from first-order logic to higher-order logic.

In [43] and [44] it was shown how to translate natural deduction proofs into expansion proofs, and an improved method of translating expansion proofs into natural deduction proofs was given. The papers [44] and [45] also contain discussions of equality and extensionality, and methods of generating more elegant natural deduction proofs.

About 1985 work started on the design and implementation of a completely new version of **TPS** to accommodate the transition from general matings to expansion proofs, take advantage of new versions of Lisp and new computers, and incorporate various improvements in the program. The initial work was done in MacLisp, yielding a program called **TPS2**, and this was later translated into Common Lisp to create the current version of **TPS**, which is called **TPS3**. **TPS3** owes a great deal to the work Dale Miller did on **TPS1** in addition to his theoretical contributions. Carl Klapper also contributed to the development of **TPS3**. Henceforth we refer to **TPS3** simply as **TPS**, since the previous versions of TPS are now obsolete.

The desirability of finding a search procedure in which expansions of the formula are motivated by the needs of the matingsearch process has long been evident. In [31] and [32] a matingsearch procedure is described in which quantifier replications are localized to vertical paths (thus reducing the enormous growth in the number of paths which accompanies replications), and the replications for each path are generated as needed to permit the construction of a mating which spans that path. The search space grows and contracts as different vertical paths are considered. This procedure has been implemented in TPS and has improved its speed very significantly.

TPS combines ideas from two fields which, regrettably, have not achieved much cross-fertilization. On the one hand, there is the "traditional" work in first-order theorem proving using such methods as resolution, model elimination, or connection graphs. On the other hand we find "avant-garde" proof-checkers and theorem provers for type theories of a variety of flavors, mostly centered around interactive proof construction with the aid of tactics.

In traditional theorem provers for first-order logic, relatively little attention has been paid to issues of human-computer interaction, but much attention has been paid to finding complete strategies which can be implemented very efficiently using, for example, advanced indexing schemes. While the use of first-order logic produces simplicity and efficiency in basic syntax and certain processes, many theorems of mathematics and other disciplines can be expressed very simply in type theory, but only in a rather complex way in first-order logic. This complexity can enormously enlarge the search space one confronts when one tries to proves these theorems.

On the other hand, tactic-based theorem provers, beginning with LCF [26] and including systems such HOL [27], Nuprl [20], the Calculus of Constructions [21], Isabelle [42] and IMPS [22], have paid considerable attention to user interaction and to the problem of formulating and supporting expressive languages for the formalization of mathematics. Techniques developed for first-order theorem proving, however, have been essentially ignored with the exception of unification, which now plays an important role in a number of these systems. Another point to note is that some of these systems chose to work in constructive logics for a variety of reasons, and that classical theorem proving techniques do not immediately apply in these circumstances. (Only recently has this gap between classical and constructive theorem proving techniques begun to close [50] [41].)

TPS unifies important ideas and concepts from both of these lines of research into a single system. It is based on classical higher-order logic, in which much of mathematics can be formalized very directly. It provides a natural deduction interface which can take advantage of the underlying theorem proving engine (the matingsearch procedure). It employs higher-order unification, finds instantiations for quantifiers on higher-order variables, and uses a machine-oriented representation of the wff for the search process. It is the combination of these features which makes TPS unique. Of course, the systems described in [11] and [17] also find proofs automatically by using techniques which have essential relevance to higher-order logic. TPS is far from comprehensive and the systems mentioned above have many other features which are not available in TPS. Perhaps closest in spirit to TPS is the work by Helmink and Ahn [28], who have also proven significant theorems in type theory (such as Cantor's theorem) completely automatically.

2. An Overview of TPS

Our experience has shown that even if one is primarily interested in the problem of proving theorems automatically, one needs good interactive tools in order to efficiently investigate examples which illuminate the fundamental logical problems of finding proofs. TPS has been designed to be a general research tool for manipulating wffs of first- and higher-order logic, and searching for proofs of such wffs interactively or automatically, or in a combination of these modes.

TPS handles two sorts of proofs:

1. *Natural deduction proofs (natural deductions).* These are human-readable (though at present boringly detailed) formal proofs. Examples are given in the figures below. In these examples we use Church's convention that a dot in a wff stands for a left bracket whose mate is as far to the right as is consistent with the pairing of brackets already present and the well-formedness of the formula. See [4] or [7] for more details about this formulation of natural deduction.

2. *Expansion proofs.* These are described briefly in [8], and studied in [36], [37], [38], [43], [44], and [45]. The structure of an expansion proof is closely and directly related to the structure of the theorem it establishes, and provides a context for search which facilitates concentrating on the essential logical structure of the theorem. At the same time, it abstracts from many details of concrete deductions. This balance between preservation of formula structure (compared to resolution refutations, for example) and abstraction of proof structure (compared to sequent derivations, for example) makes expansion proofs universal structures for cut-free (or normal) proofs. Wallen [50] provides further evidence for this by showing how expansion proofs can be adapted naturally to non-classical logics. Despite their many advantages, expansion proofs have a severe deficiency in that they are distant from formats which can be used effectively by humans.

TPS has facilities for searching for expansion proofs automatically or interactively, translating these into natural deduction proofs, constructing natural deduction proofs interactively, translating natural deduction proofs which are in normal form into expansion proofs, and solving unification problems in higher-order logic, as well as a variety of utilities designed to facilitate research and efficient interaction with the program.

The ability to translate between expansion proofs and natural deduction proofs is one of the important and attractive features of TPS. It permits both humans and computers to work in contexts which are appropriate to them. Also, we are much more confident that TPS has correctly proved a theorem when it presents us with a proof in natural deduction style than we would be if it simply indicated that it had found an expansion proof.

TPS has a number of top levels, each with its own commands. The main top level is for constructing natural deduction proofs, and there are others devoted to matings and expansion proofs and to higher-order unification problems. Another top level is a formula editor which facilitates constructing new wffs from others already known to TPS. There are editor commands for λ-conversion, Skolemization, transforming to normal forms, expanding abbreviations, counting vertical and horizontal paths, and many other manipulations of wffs. When one enters the editor, windows display the formula being edited and the particular part of the wff one is focused on.

Many aspects of the program's behavior can be controlled by setting flags, and there are over

250 of these flags. TPS has a top level called Review for examining and changing the settings of flags, and for defining and reusing groups of flag settings called *modes*.

Still another top level is a library facility for saving and displaying wffs, definitions, modes, and disagreement pairs for higher-order unification problems. Definitions can be polymorphic (i.e., contain type variables), and can contain other definitions to any level of nesting. When TPS retrieves a definition or theorem, it retrieves all the necessary subsidiary definitions. When TPS finishes proving a theorem, information about the heuristics used and statistics about the search can be stored automatically with the theorem in the library.

Yet another top level, called Test, permits one to set up experiments in which TPS will automatically try to prove a theorem a number of times, with different modes for each run, and record which mode produces the quickest proof. This top level is still rather unsophisticated.

TPS uses a type inference mechanism based on an algorithm by Milner [39] as modified by Dan Leivant. For example, we supply the following description of a wff: "a subset b and f(DB)[g x(G)] in a implies b union c = d". The notation f(DB) means that f has type $(\delta\beta)$, i.e., that it is a function mapping objects of type β to objects of type δ. Similarly, the notation x(G) means that x has type γ. From this information TPS determines the types of the function g and the sets a, b, c, and d, and (if one is using the X11 window system on one's workstation) prints the wff as

$$a_{o\delta} \subseteq b_{o\delta} \wedge f_{\delta\beta}\,[g_{\beta\gamma}\, x_{\gamma}] \in a \supset b \cup c_{o\delta} = d_{o\delta}.$$

TPS understands various conventions for omitting brackets, but by changing a suitable flag one can make TPS print the wff above as

$$[[a_{o\delta} \subseteq b_{o\delta}] \wedge [[f_{\delta\beta}\,[g_{\beta\gamma}\, x_{\gamma}]] \in a]] \supset [[b \cup c_{o\delta}] = d_{o\delta}].$$

One can also eliminate the display of type symbols (which is particularly appropriate when displaying wffs of first-order logic).

When one wishes TPS to prove a theorem automatically, one presents the theorem to TPS in the readable form illustrated above (and in the examples in section 6), and all the processing necessary to put the theorem into the form used by the search process is done automatically.

TPS can display wffs in the two-dimensional format (called a vpform) which was introduced in [5] to help one visualize the vertical paths through the wff. Examples are given in Figures 5-3 and 5-4 below.

Proofs in natural deduction style can be printed in files which are processed by Scribe or Tex, so that familiar notations of logic appear in printed proofs as well as in wffs displayed on the screen.

When one is working to construct a proof interactively, the proof can be displayed in a window called a proofwindow, which is updated automatically whenever a command which changes the proof is executed. Another proofwindow displays only the "active lines" of the proof, so that one can concentrate on the essentials of the problem. One can work forwards, backwards, or in a combination of these modes, and one can easily rearrange proofs and delete parts of proofs. One can save complete or incomplete proofs in files, and read them in at another time to continue work. One can also save the entire sequence of commands one has executed, and re-execute them later.

When one is trying to understand someone else's natural-deduction proof intuitively, it is sometimes more informative to watch the proof being constructed (working backwards and forwards at appropriate times) than to read the finished proof. One can arrange to have TPS translate an expansion proof into a natural deduction proof one step at a time, each prompted by the user, so that one can watch the natural deduction proof growing in the proofwindows at one's leisure.

Online help is available for all commands, as well as for their arguments. Considerable documentation [9], [10], [14], [33], [40], [46] has been written, though more is needed. The Facilities Guides are produced automatically.

For a number of years the purely interactive facilities of TPS have been used under the name ETPS (Educational Theorem Proving System) by students in logic courses at Carnegie Mellon to construct natural deduction proofs. ETPS permits students to concentrate on basic decisions about applying rules of inference while constructing formal proofs, gives them immediate feedback for both correct and incorrect actions, and relieves them of many of the trivial and burdensome details of writing proofs. After reviewing eight programs which support the teaching of logic, the authors of [25] (which was partially reprinted in [24]) concluded "For elementary and advanced courses in mathematical logic for students with a formal background, we choose ETPS, a powerful tool that is also easy to learn."

If the teacher of a course using ETPS wishes to use a set of rules of inference which is different from the set which comes with the program, it is quite easy to do this by using what is called the RULES module of TPS. One simply describes the new rules in a simple lisp meta-language (using the existing rules as examples), and uses commands in TPS to create the lisp code for executing these commands. (Of course, much more work would be required if one also wished to be able to automatically translate expansion proofs into natural deductions using different rules.)

TPS is a large program whose uncompiled source code contains more than 103,000 lines (including comments) and occupies about 3.75 megabytes. It is portable, runs in a variety of implementations of Common Lisp, and has been distributed to a number of researchers.

3. Tactics and Proof Translations

The basic tools in TPS for automatically applying rules of inference to construct natural deductions are *tactics*, which can be combined using *tacticals* [26]. A tactic examines a given goal situation (the problem of deriving a conclusion from a set of assumptions) and reduces it to the problem of solving a number of subgoals. This is done by applying rules of inference (forward or backwards) to derive new proof lines or to justify certain lines of the proof while introducing other lines which may still require justification.

One can use tactics to speed up the process of constructing proofs interactively. TPS has a command called GO2 which calls a number of tactics to apply mundane rules of inference to construct the easy parts of the proof, and quickly bring one to the point where some judgement and insight are needed. The user can choose whether or not to be prompted for approval before each of these tactics is applied.

The main use of tactics in TPS is to translate an expansion proof into a natural deduction by the methods of [44]; in this context, the tactics can consult the expansion proof for useful information through a number of predefined functions. We give two paradigmatic examples of such functions.

When proving a goal $[\mathbf{A} \vee \mathbf{B}]$ from some assumptions, we may need to consult the expansion proof to determine if $\mathbf{A}$ by itself already follows from the assumptions. If this is the case, we can apply disjunction introduction on the left. If not, the "disj-left" tactic fails, i.e., it does not apply. If it and its dual "disj-right" both fail, we probably want to defer a decision on the goal and try to reason forwards from the assumptions. For example, when proving $[A \vee B]$ from $[B \vee A]$, we need to distinguish the two cases (either B or A), before we can proceed with the disjunction introduction in the two subproofs. Expansion proofs are crucial in determining such information, as in general it is undecidable whether A or B follows directly. Other proof formats may also contain enough information to answer such questions, but in many cases it is obscured by preprocessing or other idiosyncrasies of various data structures devised for search.

Another example is a goal of the form $\exists x A$. In this case, we can determine from the expansion proof whether there is a substitution term t for x such that $[t/x]A$ is provable from the current set of assumptions. If so, we can derive the goal by existential generalization; otherwise, we might have to postpone application of this rule. In some cases, the expansion proof might even indicate that only the rule of indirect proof will make progress.

Basic tactics may check conditions on the expansion proof and apply appropriate inference rules. They can be combined with other tactics in different ways which can lead to different styles of proof construction. The current TPS system contains a number of basic styles which can additionally be modified through flags. The styles differ in their preference for certain inference rules and in the granularity of the rules applied. For example, one tactic applies Rule P [7], which uses arbitrary propositional tautologies, while another uses only simple rules of inference. Proofs constructed using the latter tactic are more appropriate for students of logic early in their education, while those constructed using the former are often more appropriate for mathematical arguments. Using the former tactic, TPS produces a one-line proof of $[[P_o \equiv Q_o] \equiv R_o] \equiv [P \equiv [Q \equiv R]]$, but when it uses the latter, it produces a proof 170 lines long.

The currently implemented tactics almost always produce natural deductions in normal form. This is the primary limitation of the current system, but it is clearly a consequence of the basic analytic structure of expansion proofs (and machine-generated proofs in general). The only exception is the application of *symmetric simplification* [45] to introduce simple variations of the law of excluded middle into the deduction.

TPS also partially implements a translation in the other direction, mapping normal natural deductions into expansion proofs. The utility of this translation is severely limited by the current restriction to normal deductions and has not been fully explored.

While considering how to translate expansion proofs into natural deductions, we have come to consider various aspects of the question: "How can we take advantage of the information in an expansion proof for a theorem A when constructing a natural deduction for A?" This emphasizes that it is the *form* of the natural deduction which is of primary concern; it is not sufficient to simply construct an arbitrary one from a given expansion proof.

Translating back and forth between natural deductions and expansion proofs can be used as a mechanism for intelligent restructuring of natural deductions. This mechanism can transform the structure of a natural deduction rather drastically. For example, the proof in Figure 3-1 was translated into an expansion proof, and then back to a natural deduction using symmetric simplification to produce the proof in Figure 3-2. The original proof consisted of a rather unintuitive, brute force, indirect proof, while the transformed proof identifies the crucial case distinction which should be made: either "P" is true everywhere or not. An even simpler proof could have been obtained using the lemma [$\forall x\ P\ x \vee \exists x\ \sim P\ x$], which is beyond the scope of our current methods. (See [45] for further discussion.)

Figure 3-1: Original proof of X2119

(1)	1	$\vdash$	$\sim\exists y\ \forall x\ .P\ y \supset P\ x$	Assume negation
(2)	1	$\vdash$	$\forall y\ .\sim\forall x\ .P\ y \supset P\ x$	Neg: 1
(3)	1	$\vdash$	$\sim\forall x\ .P\ y^1 \supset P\ x$	UI: y^1 2
(4)	1	$\vdash$	$\exists x\ .\sim.P\ y^1 \supset P\ x$	Neg: 3
(5)	1	$\vdash$	$\sim\forall x\ .P\ y^2 \supset P\ x$	UI: y^2 2
(6)	1,6	$\vdash$	$\sim.P\ y^1 \supset P\ y^2$	Choose: y^2
(7)	1	$\vdash$	$\exists x\ .\sim.P\ y^2 \supset P\ x$	Neg: 5
(8)	1,6,8	$\vdash$	$\sim.P\ y^2 \supset P\ y^3$	Choose: y^3
(9)	1,6	$\vdash$	$P\ y^1 \wedge \sim P\ y^2$	Neg: 6
(10)	1,6	$\vdash$	$P\ y^1$	Conj: 9
(11)	1,6	$\vdash$	$\sim P\ y^2$	Conj: 9
(12)	1,6,8	$\vdash$	$P\ y^2 \wedge \sim P\ y^3$	Neg: 8
(13)	1,6,8	$\vdash$	$P\ y^2$	Conj: 12
(14)	1,6,8	$\vdash$	$\sim P\ y^3$	Conj: 12
(15)	1,6,8	$\vdash$	$\perp$	RuleP: 11 13
(16)	1,6	$\vdash$	$\perp$	RuleC: 7 15
(17)	1	$\vdash$	$\perp$	RuleC: 4 16
(18)		$\vdash$	$\exists y\ \forall x\ .P\ y \supset P\ x$	Indirect: 17

Figure 3-2: Transformed proof of X2119

(1)		$\vdash$	$\forall x\ P\ x \vee \sim\forall x\ P\ x$	RuleP
(2)	2	$\vdash$	$\forall x\ P\ x$	Case 1: 1
(3)	2	$\vdash$	$P\ w$	UI: w 2
(4)	2	$\vdash$	$P\ y^1 \supset P\ w$	Deduct: 3
(5)	2	$\vdash$	$\forall w\ .P\ y^1 \supset P\ w$	UGen: w 4
(6)	2	$\vdash$	$\forall x\ .P\ y^1 \supset P\ x$	AB: 5
(7)	2	$\vdash$	$\exists y\ \forall x\ .P\ y \supset P\ x$	EGen: y^1 6
(8)	8	$\vdash$	$\sim\forall x\ P\ x$	Case 2: 1
(9)	9	$\vdash$	$\sim\exists y\ \forall x\ .P\ y \supset P\ x$	Assume negation
(10)	10	$\vdash$	$\sim P\ y^2$	Assume negation
(11)	10	$\vdash$	$P\ y^2 \supset P\ x$	RuleP: 10
(12)	10	$\vdash$	$\forall x\ .P\ y^2 \supset P\ x$	UGen: x 11
(13)	10	$\vdash$	$\exists y\ \forall x\ .P\ y \supset P\ x$	EGen: y^2 12
(14)	9,10	$\vdash$	$\perp$	NegElim: 9 13
(15)	9	$\vdash$	$P\ y^2$	Indirect: 14
(16)	9	$\vdash$	$\forall y^2\ P\ y^2$	UGen: y^2 15
(17)	9	$\vdash$	$\forall x\ P\ x$	AB: 16
(18)	8,9	$\vdash$	$\perp$	NegElim: 8 17
(19)	8	$\vdash$	$\exists y\ \forall x\ .P\ y \supset P\ x$	Indirect: 18
(20)		$\vdash$	$\exists y\ \forall x\ .P\ y \supset P\ x$	Cases: 1 7 19

The equality relation is ubiquitous in mathematical reasoning, and special mechanisms for dealing with equality, such as those in [23] and [49], are clearly needed. At present, TPS has no such mechanisms, and simply defines equality by the Leibniz definition $[\lambda x_\alpha\ \lambda y_\alpha\ \forall q_{o\alpha} .q\ x \supset q\ y]$ or by the extensional definition $[\lambda x_{\alpha\beta}\ \lambda y_{\alpha\beta}\ \forall z_\beta .x\ z = y\ z]$ for equality between functions or sets, and uses ordinary laws of logic to prove results involving equality. An example of this is in Figure 3-3, where we show the main part of a proof constructed automatically for THM104. (This theorem and the definition of *U* are discussed further in section 6). In [44] it was shown how an expansion proof for a theorem involving equality can be translated into a natural deduction containing traditional equality inferences, and this has been implemented in TPS. The application of this feature is controlled by the flag REMOVE-LEIBNIZ. Thus, when we change the value of this flag from NIL to T, the same expansion proof which generated the natural deduction in Figure 3-3 generates the one in Figure 3-4.

Figure 3-3: Main part of proof of THM104 with REMOVE-LEIBNIZ set to NIL

(1)	1	$\vdash$	$U\ X_\alpha = U\ Z_\alpha$	Hyp
(2)	1	$\vdash$	$\lambda y_\alpha\ [X_\alpha = y] = \lambda y\ .Z_\alpha = y$	EquivWffs: 1
(3)	1	$\vdash$	$\forall q_{o(o\alpha)}\ .q\ [\lambda y_\alpha\ .X_\alpha = y] \supset q\ .\lambda y\ .Z_\alpha = y$	Equality: 2
(5)	1	$\vdash$	$[\lambda w_{o\alpha}\ .w\ X_\alpha]\ [\lambda y_\alpha\ .X = y] \supset [\lambda w\ .w\ X]\ .\lambda y\ .Z_\alpha = y$	UI: $[\lambda w_{o\alpha}\ .w\ X_\alpha]$ 3
(6)	1	$\vdash$	$X_\alpha = X \supset Z_\alpha = X$	Lambda: 5
(7)		$\vdash$	$q_{o\alpha}\ X_\alpha \supset q\ X$	RuleP
(8)		$\vdash$	$\forall q_{o\alpha}\ .q\ X_\alpha \supset q\ X$	UGen: $q_{o\alpha}$ 7
(9)		$\vdash$	$X_\alpha = X$	Equality: 8
(10)	1	$\vdash$	$Z_\alpha = X_\alpha$	MP: 9 6

Figure 3-4: Main part of proof of THM104 with REMOVE-LEIBNIZ set to T

(1)	1	$\vdash$	$U\ X_\alpha = U\ Z_\alpha$	Hyp
(2)	1	$\vdash$	$\lambda y_\alpha\ [X_\alpha = y] = \lambda y\ .Z_\alpha = y$	EquivWffs: 1
(3)		$\vdash$	$X_\alpha = X$	Assert REFL=
(4)		$\vdash$	$[\lambda y_\alpha\ .X_\alpha = y]\ X$	Lambda: 3
(5)	1	$\vdash$	$[\lambda y_\alpha\ .Z_\alpha = y]\ X_\alpha$	Subst=: 4 2
(6)	1	$\vdash$	$Z_\alpha = X_\alpha$	Lambda: 5

4. Automatic Search

When one asks TPS to find a proof for a theorem automatically, it starts out by searching for an expansion proof of the theorem, and then translates this into a natural deduction proof. It sets up an expansion tree to represent the wff, and searches for an acceptable mating [5] of its literals. (An expansion tree which is appropriately expanded and has an acceptable mating is an expansion proof.)

The search process is controlled by a number of flags. Ideally, TPS would have heuristics to decide how to set these flags, but at present the user does this interactively before starting the automatic search. The flags provide a convenient way to explore many different aspects of the problem of searching for proofs.

When one is seeking an expansion proof for a theorem of higher-order logic, not all necessary substitution terms can be generated by unification of formulas already present, so certain *expansion options* [8] are applied to the expansion tree which represents the theorem. In [8] we discussed expansion options which consisted of applying *primitive substitutions* to predicate variables. These substitutions introduce a single quantifier or connective, and contain variables for which additional substitutions can be made at a later stage. However, we do not yet have good heuristics to guide the process of applying primitive substitutions incrementally, so we currently use a procedure which introduces substitution terms containing more logical structure, and then searches for a mating without making further substitutions for the variables in the substitution terms except as dictated by the unifier associated with the mating. In order to limit the number of forms which must be considered for the substitution terms, we often use (in addition to projections) terms whose bodies are in prenex normal form with matrix in conjunctive or disjunctive normal form. We call these substitutions *gensubs* (general substitutions). Primitive substitutions are special cases of gensubs. The logical complexity of the gensubs which TPS will attempt to apply is limited by a flag called MAX-PRIM-DEPTH. Examples of gensubs for $r_{o\beta(o\beta)}$ are given in Figure 4-1. The gensubs for a variable are determined (up to renaming of auxiliary variables) by the type of the variable. Since a gensub substitutes a term for just one variable, we shall sometimes use the name of the gensub to denote the substitution term. The types of quantified variables in gensubs (such as β in the figure) are chosen from a small fixed set of types which is specified as the value of the flag PRIM-BDTYPES. One can permit TPS to apply a rather naive algorithm for choosing this set of types by setting another flag, called PRIM-BDTYPES-AUTO.

Different sets of expansion options are applied to create different expansion trees which are all subtrees of a master expansion tree. The sets of expansion options are generated in a systematic and exhaustive way whose details are determined by certain flags. Thus a potentially infinite list of subtrees is generated; smaller subtrees are explored before larger ones in an attempt to keep the search space manageable. Of course, this blind generation of sets of expansion options is rather crude. We look forward to the development of heuristics and metatheorems to improve the sophistication of this process by guiding or restricting it.

Before searching for an acceptable mating of the literals in a subtree, TPS converts the subtree into an alternative representation called a *jform* (junctive form). If the connectives truth and falsehood occur, they are eliminated, with appropriate adjustments in the jform corresponding to the relevant laws of propositional calculus. Then TPS tries to build up a mating which spans every vertical path through the jform by progressively adding links to span the paths. When working on higher-order theorems, it uses Huet's higher-order unification algorithm [30] to check the compatibility of the connections in the partial mating. A unification tree is associated with the partial mating, and when a new connection is added to the mating, the associated disagreement pair is added to all the leaves of the unification tree. When incompatibilities are encountered, the process backtracks. Since higher-order unification may not terminate, TPS is not permitted to generate nodes of the unification tree with depth greater than the value of the flag MAX-SEARCH-DEPTH. (If this flag is set to a high value, TPS will often spend an enormous amount of time generating higher-order unification search trees.) The search for an acceptable mating within a given jform may not terminate, so when the time

Figure 4-1: Gensubs for $r_{o\beta(o\beta)}$

Gensubs with MAX-PRIM-DEPTH 2:

P0 $\lambda w^1_{o\beta}\ \lambda w^2_{\beta} . r^1_{o\beta(o\beta)}\ w^1\ w^2 \land r^2_{o\beta(o\beta)}\ w^1\ w^2$

P1 $\lambda w^1_{o\beta}\ \lambda w^2_{\beta} . r^3_{o\beta(o\beta)}\ w^1\ w^2 \lor r^4_{o\beta(o\beta)}\ w^1\ w^2$

P2 $\lambda w^1_{o\beta}\ \lambda w^2_{\beta}\ w^1 . r^5_{\beta\beta(o\beta)}\ w^1\ w^2$

P3 $\lambda w^1_{o\beta}\ \lambda w^2_{\beta}\ \exists w^3_{\beta}\ r^6_{o\beta\beta(o\beta)}\ w^1\ w^2\ w^3$

P4 $\lambda w^1_{o\beta}\ \lambda w^2_{\beta}\ \forall w^3_{\beta}\ r^7_{o\beta\beta(o\beta)}\ w^1\ w^2\ w^3$

P5 $\lambda w^1_{o\beta}\ \lambda w^2_{\beta}\ \exists w^4_{\beta} . r^8_{o\beta\beta(o\beta)}\ w^1\ w^2\ w^4 \lor r^9_{o\beta\beta(o\beta)}\ w^1\ w^2\ w^4$

P6 $\lambda w^1_{o\beta}\ \lambda w^2_{\beta}\ \exists w^4_{\beta} . r^{10}_{o\beta\beta(o\beta)}\ w^1\ w^2\ w^4 \land r^{11}_{o\beta\beta(o\beta)}\ w^1\ w^2\ w^4$

P7 $\lambda w^1_{o\beta}\ \lambda w^2_{\beta}\ \forall w^4_{\beta} . r^{12}_{o\beta\beta(o\beta)}\ w^1\ w^2\ w^4 \lor r^{13}_{o\beta\beta(o\beta)}\ w^1\ w^2\ w^4$

P8 $\lambda w^1_{o\beta}\ \lambda w^2_{\beta}\ \forall w^4_{\beta} . r^{14}_{o\beta\beta(o\beta)}\ w^1\ w^2\ w^4 \land r^{15}_{o\beta\beta(o\beta)}\ w^1\ w^2\ w^4$

Examples of additional gensubs with MAX-PRIM-DEPTH 3:

P13 $\lambda w^1_{o\beta}\ \lambda w^2_{\beta}\ \exists w^5_{\beta}\ \forall w^6_{\beta} .$
$[r^{40}_{o\beta\beta\beta(o\beta)}\ w^1\ w^2\ w^5\ w^6 \land r^{41}_{o\beta\beta\beta(o\beta)}\ w^1\ w^2\ w^5\ w^6]$
$\lor\ [r^{42}_{o\beta\beta\beta(o\beta)}\ w^1\ w^2\ w^5\ w^6 \land r^{43}_{o\beta\beta\beta(o\beta)}\ w^1\ w^2\ w^5\ w^6]$
$\lor\ [r^{44}_{o\beta\beta\beta(o\beta)}\ w^1\ w^2\ w^5\ w^6 \land r^{45}_{o\beta\beta\beta(o\beta)}\ w^1\ w^2\ w^5\ w^6]$

P16 $\lambda w^1_{o\beta}\ \lambda w^2_{\beta}\ \forall w^5_{\beta}\ \forall w^6_{\beta} .$
$[r^{58}_{o\beta\beta\beta(o\beta)}\ w^1\ w^2\ w^5\ w^6 \lor r^{59}_{o\beta\beta\beta(o\beta)}\ w^1\ w^2\ w^5\ w^6]$
$\land\ [r^{60}_{o\beta\beta\beta(o\beta)}\ w^1\ w^2\ w^5\ w^6 \lor r^{61}_{o\beta\beta\beta(o\beta)}\ w^1\ w^2\ w^5\ w^6]$
$\land\ [r^{62}_{o\beta\beta\beta(o\beta)}\ w^1\ w^2\ w^5\ w^6 \lor r^{63}_{o\beta\beta\beta(o\beta)}\ w^1\ w^2\ w^5\ w^6]$

---

spent on it reaches the value of the flag SEARCH-TIME-LIMIT, TPS temporarily abandons this jform and tries another one. It may return to that jform later, but there is also a limit (specified by the flag MAX-SEARCH-LIMIT) on the total amount of time which can be spent on searching for a proof in any jform. Once an acceptable mating is found, it is converted into an expansion proof, which is simplified by a process called *merging*, and then translated into a natural deduction proof.

Now that we have outlined the general procedure TPS uses to find a proof, let us illustrate the

Figure 4-2: Outline of proof of X5310

(1) 1 $\vdash$ $\forall r_{o\beta(o\beta)} .\forall p_{o\beta}\ \exists x_{\beta}\ r\ p\ x \supset \exists j_{\beta(o\beta)}\ \forall p\ r\ p\ .j\ p$ Hyp

(2) 1 $\vdash$ $\forall p_{o\beta}\ \exists x_{\beta}\ [\lambda p\ \lambda y_{\beta}\ .\exists x\ p\ x \supset p\ y]\ p\ x$
$\supset \exists j_{\beta(o\beta)}\ \forall p\ [\lambda p\ \lambda y\ .\exists x\ p\ x \supset p\ y]\ p\ .j\ p$
UI: $[\lambda p_{o\beta}\ \lambda y_{\beta}.\exists x_{\beta}\ p\ x \supset p\ y]$ 1

(3) 1 $\vdash$ $\forall p_{o\beta}\ \exists x_{\beta}\ [\exists x\ p\ x \supset p\ x] \supset \exists j_{\beta(o\beta)}\ \forall p\ .\exists x\ p\ x \supset p.j\ p$
Lambda: 2

...

(98) 1 $\vdash$ $\exists j_{\beta(o\beta)}\ \forall p_{o\beta}\ .\exists x_{\beta}\ p\ x \supset p\ .j\ p$ PLAN3

(99) $\vdash$ $\forall r_{o\beta(o\beta)}\ [\forall p_{o\beta}\ \exists x_{\beta}\ r\ p\ x \supset \exists j_{\beta(o\beta)}\ \forall p\ r\ p\ .j\ p]$
$\supset \exists j\ \forall p\ .\exists x\ p\ x \supset p\ .j\ p$ Deduct: 98

(100) $\vdash$ $\forall r_{o\beta(o\beta)}\ [\forall x_{o\beta}\ \exists y_{\beta}\ r\ x\ y \supset \exists f_{\beta(o\beta)}\ \forall x\ r\ x\ .f\ x]$
$\supset \exists j_{\beta(o\beta)}\ \forall p_{o\beta}\ .\exists x_{\beta}\ p\ x \supset p\ .j\ p$ AB: 99

Figure 4-3: Substitution terms for $r_{o\beta(o\beta)}$ for proof of X5310

Original gensub:

P7 $[\lambda w^1{}_{o\beta}\ \lambda w^2{}_{\beta}\ \forall w^4{}_{\beta}.r^{12}{}_{o\beta\beta(o\beta)}\ w^1\ w^2\ w^4 \lor r^{13}{}_{o\beta\beta(o\beta)}\ w^1\ w^2\ w^4]$

Result of substituting for the free variables of P7:

M $[\lambda w^1{}_{o\beta}\ \lambda w^2{}_{\beta}\ \forall w^4{}_{\beta}.w^1\ w^2 \lor \sim w^1\ w^4]$

Alphabetic variant of M:

M1 $[\lambda p_{o\beta}\ \lambda y_{\beta}\ \forall x_{\beta}.p\ y \lor \sim p\ x]$

Crucial substitution term in Figure 4-2:

N $[\lambda p_{o\beta}\ \lambda y_{\beta}.\exists x_{\beta}\ p\ x \supset p\ y]$

use of gensubs by discussing how **TPS** proves theorem X5310, which may be found at the end of section 6. In order to understand X5310 the reader is advised to first look at the companion theorem X5308. The outline of a simple proof of X5310 is in Figure 4-2. The theorem to be proved is in line (100), but the alphabetic variant of it in (99) is easier to work with. The problem is to derive (98) (which is the Axiom of Choice) from the hypothesis in (1). The key step is to instantiate the quantifier $\forall r_{o\beta(o\beta)}$ in (1) with the term $[\lambda p_{o\beta}\ \lambda y_{\beta}.\exists x_{\beta}\ p\ x \supset p\ y]$, which we shall call N. Doing this yields (3), whose antecedent is easily provable, and whose consequent is (98).

In the search for a proof, **TPS** eventually considers the set of expansion options which simply substitutes gensub P7 of Figure 4-1 (which is also displayed in Figure 4-3) for $r_{o\beta(o\beta)}$. **TPS** finds an acceptable mating of the associated jform, and applies the unifier associated with the mating to the free variables $r^{12}{}_{o\beta\beta(o\beta)}$ and $r^{13}{}_{o\beta\beta(o\beta)}$ of P7 to obtain the wff M of Figure 4-3; an alphabetic variant of this is the term with which **TPS** instantiates $\forall r_{o\beta(o\beta)}$ in the natural deduction proof. For convenience we display the alphabetic variant M1 of M, which can be transformed to the wff N mentioned above by applying elementary logical equivalences.

Note that while the quantified variable in P7 occurs in both the left and the right scopes of the disjunction, this is not the case for M1. Thus the gensub P7 is truly a general wff which can take several forms, one of which is M.

While the use of prenex normal forms in this context drastically reduces the number of sets of expansion options which must be considered, and seems not to complicate the search for an acceptable mating as long as the quantifiers thus introduced need not be duplicated, one does pay a price for using the prenex formula M instead of the miniscope formula N. The natural deduction proof for X5310 which TPS constructs is 57 lines long and is rather clumsy. It may be possible to remedy this by modifying the procedure so that it repeats part of the process after the correct instantiation terms have been discovered and put into miniscope form.

As noted in [8], restricting expansion terms to some normal form may (depending on other details of the implementation) entail loss of completeness of a proof procedure, but for the present we are content to explore the benefits of using gensubs. Questions related to combining this method of instantiating quantifiers on higher-order variables with other methods, such as those of [11], [16], and [17], need further study. Some of the examples found in these papers are discussed below.

TPS can duplicate quantifiers during the search for a mating by using outermost-quantifier duplication [5] or by using path-focused duplication [31]. It can also generate sets of expansion options in several ways, and there are several implementations of both first- and higher-order unification algorithms. These basic facilities are combined into a number of search procedures in TPS. Much remains to be done in exploring the relative merits of various search techniques in various situations, and strategies for systematically incrementing the flags which control the size of the multi-dimensional search space.

Some of the search procedures in TPS take into account the fact that for any subformula of the form $[A \vee B]$, any path which passes through A has a variant which passes through B, and both of these paths must be spanned by an acceptable mating. Therefore, when the matingsearch procedure comes to B, if A has no mate, then no mate for B will be sought. Also, if A has a mate, but no mate for B can be found, then the search will backtrack, throwing out the mate for A and all links which were subsequently added to the mating.

Since the search procedures used by TPS treat the paths very systematically (and unimaginatively) in their natural order, TPS requires very little space to keep track of what it has done. (This may be contrasted, for example, with resolution systems which store vast numbers of clauses). The higher-order unification procedure does introduce many auxiliary variables, but they are used only briefly, and it was found that by simply uninterning them a great deal of space could be reclaimed. Consequently, TPS can run for weeks without running out of space (particularly on problems of first-order logic). However, for many theorems one clearly needs to use more flexible heuristically-guided search procedures. It is a significant problem to find a good balance between applying sophisticated search heuristics and limiting the amount of memory required to keep track of the process of exploring the search space.

Following (and slightly extending) terminology introduced by Huet [30], we refer to literals whose atoms are headed by predicate variables as *flexible*, and to literals whose atoms are headed by predicate constants as *rigid*. In first-order logic all predicate symbols may be regarded as constants, so flexible literals need not be considered. However, they do frequently occur in higher-order logic, for example when the Leibniz definition of equality is instantiated. Applications of gensubs to the head variables of flexible literals create even more such literals. Since flexible literals can be mated with arbitrary literals, the search space associated with finding an acceptable mating for a wff which contains many such literals can be extremely large. Two heuristics, each controlled by a flag, are available in TPS for trying to cope with this problem. First, the user can set the flag MAX-MATES to limit the number of mates which any literal-occurrence may have. (It is fairly rare to encounter examples of acceptable matings in which literals have many mates.) Since the algorithm for constructing matings in TPS tries to span each path by mating the first available pair of literals, and a flexible literal typically occurs on many paths, this limit quickly excludes from consideration many matings which the unification process would find incompatible only after considerable work. Second, one can rearrange the jform before the search for a mating commences. If the user sets the flag ORDER-COMPONENTS to the value PREFER-RIGID1, TPS applies an algorithm to rearrange the jform (using the commutativity and associativity of conjunction and disjunction) so that rigid literals tend to be encountered by the search process before flexible literals; this postpones finding mates for flexible literals until constraints introduced by mating other literals have been introduced.

Additional complexity arises when one mates a pair of flexible literals. Mating such a pair can cause either literal to become the negation of the other (after substitution, λ-reduction, and elimination of double negations). Both possibilities must be considered, since the variables in these literals may occur in other literals too. When TPS considers adding such a pair to the mating, it puts a disagreement pair corresponding to it into the leaves of the unification tree, and proceeds with the unification process. If this process encounters a disagreement pair of the form <**A**, ~**B**>, where **A** starts with a constant but **B** does not, it replaces this pair with <~**A**, **B**> and continues. Thus, in a very economical fashion TPS finds which way of mating these literals is ultimately compatible with the unification problem for the entire mating.

If one is working on a theorem which is too hard for TPS to prove all by itself, one can still use the automatic facilities of TPS to provide assistance. One can use the interactive facilities of TPS (supplemented by GO2) to develop the general outline of the proof in natural deduction style, and ask TPS to help by automatically proving certain lines of the proof from other specified lines.

5. An Example

One of the most interesting theorems which TPS has proved automatically is THM15B:
$\forall f_{\iota\iota}.\exists g_{\iota\iota}\, [\text{ITERATE+}\ f\ g \wedge \exists x_{\iota}.g\ x = x \wedge \forall z_{\iota}.g\ z = z \supset z = x] \supset \exists y_{\iota}.f\ y = y$
TPS takes about 2.5 hours to prove this theorem, which asserts that if some iterate of a function f has a unique fixed point, then f has a fixed point. (The definition of ITERATE+ is in section 6.) The theorem appears in [34], where it is pointed out that the theorem is useful in the theory of functional equations. It was posed as a problem for theorem provers in [1], and it is gratifying that we are finally able to prove it automatically. It is a hard theorem for TPS because so many flexible literals are created when the definition of equality is instantiated.

The very natural though overly detailed proof which TPS finds is shown in Figures 5-1 and 5-2. Let us summarize it by providing comments for some of the lines of the proof. TPS starts out by

Figure 5-1: Proof of THM15B - part 1

(1)	1	$\vdash$	$\exists g_{\iota\iota}\ .\ \text{ITERATE+}\ f_{\iota\iota}\ g$	
			$\wedge\ \exists x_\iota\ .g\ x = x \wedge \forall z_\iota\ .g\ z = z \supset z = x$	Hyp
(2)	1,2	$\vdash$	$\text{ITERATE+}\ f_{\iota\iota}\ g_{\iota\iota} \wedge \exists x_\iota\ .g\ x = x \wedge \forall z_\iota\ .g\ z = z \supset z = x$	
				Choose: $g_{\iota\iota}$ 1
(3)	1,2	$\vdash$	$\text{ITERATE+}\ f_{\iota\iota}\ g_{\iota\iota}$	RuleP: 2
(4)	1,2	$\vdash$	$\exists x_\iota\ .g_{\iota\iota}\ x = x \wedge \forall z_\iota\ .g\ z = z \supset z = x$	RuleP: 2
(5)	1,2	$\vdash$	$\forall p_{o(\iota\iota)}\ .p\ f_{\iota\iota} \wedge \forall j_{\iota\iota}\ [p\ j \supset p\ .\lambda x_\iota\ f\ .j\ x] \supset p\ g_{\iota\iota}$	
				EquivWffs: 3
(6)	1,2,6	$\vdash$	$g_{\iota\iota}\ x_\iota = x \wedge \forall z_\iota\ .g\ z = z \supset z = x$	Choose: x_ι 4
(7)	1,2,6	$\vdash$	$g_{\iota\iota}\ x_\iota = x$	RuleP: 6
(8)	1,2,6	$\vdash$	$\forall z_\iota\ .g_{\iota\iota}\ z = z \supset z = x_\iota$	RuleP: 6
(9)	1,2,6	$\vdash$	$g_{\iota\iota}\ [f_{\iota\iota}\ x_\iota] = f\ x \supset f\ x = x$	UI: $[f_{\iota\iota}\ x_\iota]$ 8
(10)	1,2	$\vdash$	$[\lambda j_{\iota\iota}\ \forall P_{o\iota}\ .P\ [f_{\iota\iota}\ .j\ x_\iota] \vee \sim P\ .j\ .f\ x]\ f$	
			$\wedge\ \forall j\ [\ [\lambda j\ \forall P\ .P\ [f\ .j\ x] \vee \sim P\ .j\ .f\ x]\ j$	
			$\supset [\lambda j\ \forall P\ .P\ [f\ .j\ x] \vee \sim P\ .j\ .f\ x]\ .\lambda x\ f\ .j\ x]$	
			$\supset [\lambda j\ \forall P\ .P\ [f\ .j\ x] \vee \sim P\ .j\ .f\ x]\ g_{\iota\iota}$	
				UI: $[\lambda j_{\iota\iota}\ \forall P_{o\iota}.P\ [f_{\iota\iota}.j\ x_\iota] \vee \sim P.j.f\ x]$ 5
(11)	1,2	$\vdash$	$\forall P_{o\iota}\ [P\ [f_{\iota\iota}\ .f\ x_\iota] \vee \sim P\ .f\ .f\ x]$	
			$\wedge\ \forall j_{\iota\iota}\ [\ \forall P\ [P\ [f\ .j\ x] \vee \sim P\ .j\ .f\ x]$	
			$\supset \forall P\ .P\ [f\ .f\ .j\ x] \vee \sim P\ .f\ .j\ .f\ x]$	
			$\supset \forall P\ .P\ [f\ .g_{\iota\iota}\ x] \vee \sim P\ .g\ .f\ x$	Lambda: 10
(12)		$\vdash$	$P_{o\iota}\ [f_{\iota\iota}\ .f\ x_\iota] \vee \sim P\ .f\ .f\ x$	RuleP
(13)		$\vdash$	$\forall P_{o\iota}\ .P\ [f_{\iota\iota}\ .f\ x_\iota] \vee \sim P\ .f\ .f\ x$	UGen: $P_{o\iota}$ 12
(14)	14	$\vdash$	$\forall P_{o\iota}\ .P\ [f_{\iota\iota}\ .j_{\iota\iota}\ x_\iota] \vee \sim P\ .j\ .f\ x$	Hyp
(15)	14	$\vdash$	$[\lambda w_\iota\ W^9_{o\iota}\ .f_{\iota\iota}\ w]\ [f\ .j_{\iota\iota}\ x_\iota] \vee \sim[\lambda w\ W^9\ .f\ w]\ .j\ .f\ x$	
				UI: $[\lambda w_\iota\ W^9_{o\iota}.f_{\iota\iota}\ w]$ 14
(16)	14	$\vdash$	$W^9_{o\iota}\ [f_{\iota\iota}\ .f\ .j_{\iota\iota}\ x_\iota] \vee \sim W^9\ .f\ .j\ .f\ x$	Lambda: 15
(17)	14	$\vdash$	$\forall W^9_{o\iota}\ .W^9\ [f_{\iota\iota}\ .f\ .j_{\iota\iota}\ x_\iota] \vee \sim W^9\ .f\ .j\ .f\ x$	UGen: $W^9_{o\iota}$ 16
(18)	14	$\vdash$	$\forall P_{o\iota}\ .P\ [f_{\iota\iota}\ .f\ .j_{\iota\iota}\ x_\iota] \vee \sim P\ .f\ .j\ .f\ x$	AB: 17
(19)		$\vdash$	$\forall P_{o\iota}\ [P\ [f_{\iota\iota}\ .j_{\iota\iota}\ x_\iota] \vee \sim P\ .j\ .f\ x]$	
			$\supset \forall P\ .P\ [f\ .f\ .j\ x] \vee \sim P\ .f\ .j\ .f\ x$	Deduct: 18
(20)		$\vdash$	$\forall j_{\iota\iota}\ .\ \forall P_{o\iota}\ [P\ [f_{\iota\iota}\ .j\ x_\iota] \vee \sim P\ .j\ .f\ x]$	
			$\supset \forall P\ .P\ [f\ .f\ .j\ x] \vee \sim P\ .f\ .j\ .f\ x$	UGen: $j_{\iota\iota}$ 19
(21)		$\vdash$	$\forall P_{o\iota}\ [P\ [f_{\iota\iota}\ .f\ x_\iota] \vee \sim P\ .f\ .f\ x]$	
			$\wedge\ \forall j_{\iota\iota}\ .\ \forall P\ [P\ [f\ .j\ x] \vee \sim P\ .j\ .f\ x]$	
			$\supset \forall P\ .P\ [f\ .f\ .j\ x] \vee \sim P\ .f\ .j\ .f\ x$	
				RuleP: 13 20
(22)	1,2	$\vdash$	$\forall P_{o\iota}\ .P\ [f_{\iota\iota}\ .g_{\iota\iota}\ x_\iota] \vee \sim P\ .g\ .f\ x$	MP: 21 11
(23)	1,2	$\vdash$	$g_{\iota\iota}\ [f_{\iota\iota}\ x_\iota] = f\ [g\ x] \vee \sim .g\ [f\ x] = g\ .f\ x$	
				UI: $[=.g_{\iota\iota}.f_{\iota\iota}\ x_\iota]$ 22
(24)	1,2,24	$\vdash$	$g_{\iota\iota}\ [f_{\iota\iota}\ x_\iota] = f\ .g\ x$	Case 1: 23

Figure 5-2: Proof of THM15B - part 2

(25)	1,2,25	$\vdash$	$\sim . g_{\iota\iota}\ [f_{\iota\iota}\ x_\iota] = g\ .f\ x$	Case 2: 23
(26)	1,2,25	$\vdash$	$\sim \mathbf{T}$	Refl=: 25
(27)	1,2,25	$\vdash$	$g_{\iota\iota}\ [f_{\iota\iota}\ x_\iota] = f\ .g\ x$	RuleP: 26
(28)	1,2	$\vdash$	$g_{\iota\iota}\ [f_{\iota\iota}\ x_\iota] = f\ .g\ x$	Cases: 23 24 27
(29)	1,2,6	$\vdash$	$g_{\iota\iota}\ [f_{\iota\iota}\ x_\iota] = f\ x$	Subst=: 28 7
(30)	1,2,6	$\vdash$	$f_{\iota\iota}\ x_\iota = x$	MP: 29 9
(31)	1,2,6	$\vdash$	$\exists y_\iota\ .f_{\iota\iota}\ y = y$	EGen: x_ι 30
(32)	1,2	$\vdash$	$\exists y_\iota\ .f_{\iota\iota}\ y = y$	RuleC: 4 31
(33)	1	$\vdash$	$\exists y_\iota\ .f_{\iota\iota}\ y = y$	RuleC: 1 32
(34)		$\vdash$	$\exists g_{\iota\iota}\ [\ \text{ITERATE+}\ f_{\iota\iota}\ g \wedge \exists x_\iota\ .g\ x = x \wedge \forall z_\iota\ .g\ z = z \supset z = x] \supset \exists y_\iota\ .f\ y = y$	Deduct: 33
(35)		$\vdash$	$\forall f_{\iota\iota}\ .\ \exists g_{\iota\iota}\ [\ \text{ITERATE+}\ f\ g \wedge \exists x_\iota\ .g\ x = x \wedge \forall z_\iota\ .g\ z = z \supset z = x] \supset \exists y_\iota\ .f\ y = y$	UGen: $f_{\iota\iota}$ 34

---

assuming that g is an iterate of f,

(3)	1,2	$\vdash$	$\text{ITERATE+}\ f_{\iota\iota}\ g_{\iota\iota}$	RuleP: 2

that x is a fixed point of g,

(7)	1,2,6	$\vdash$	$g_{\iota\iota}\ x_\iota = x$	RuleP: 6

and that x is the only fixed point of g.

(8)	1,2,6	$\vdash$	$\forall z_\iota\ .g_{\iota\iota}\ z = z \supset z = x_\iota$	RuleP: 6

In lines (5) and (10)-(28) TPS then gives an inductive proof, based on the definition in (3), of the fact that

(28)	1,2	$\vdash$	$g_{\iota\iota}\ [f_{\iota\iota}\ x_\iota] = f\ .g\ x$	Cases: 23 24 27

Of course, (28) follows from the fact that f commutes with all its iterates. No knowledge about commutativity or iterates is built into TPS except the definition of ITERATE+. TPS decides to prove (28) simply by applying general logical principles in its search for an expansion proof of the theorem. From (7) and (28) TPS concludes that

(29)	1,2,6	$\vdash$	$g_{\iota\iota}\ [f_{\iota\iota}\ x_\iota] = f\ x$	Subst=: 28 7

Line (29) shows that f x is also a fixed point of g, so it must be the same as x:

(9)	1,2,6	$\vdash$	$g_{\iota\iota}\ [f_{\iota\iota}\ x_\iota] = f\ x \supset f\ x = x$	UI: $[f_{\iota\iota}\ x_\iota]$ 8
(30)	1,2,6	$\vdash$	$f_{\iota\iota}\ x_\iota = x$	MP: 29 9

Thus x is a fixed point of f.

It is natural to wonder how such a proof can be found, so let us see how TPS does this. In the preprocessing stage, it negates the theorem and eliminates the definitions, obtaining:

$$\begin{aligned}
&\exists f_\iota.\exists g_\iota\ [\forall p_{o(\iota)}\ [p\ f \wedge \forall j_\iota\ [p\ j \supset p.\lambda x_\iota\ f.j\ x] \supset p\ g] \\
&\wedge \exists x.g\ x = x \\
&\wedge \forall z_\iota.\sim[g\ z = z] \vee z = x] \\
&\wedge \forall y_\iota.\sim.f\ y = y
\end{aligned}$$

It then expands the equality formulas using the Leibniz definition, and puts the wff into negation normal form, obtaining:

$$\begin{aligned}
&\exists f_\iota.\exists g_\iota\ [\forall p_{o(\iota)}\ [\sim p\ f \vee \exists j_\iota\ [p\ j \wedge \sim p.\lambda x_\iota\ f.j\ x] \vee p\ g] \\
&\wedge \exists x.\forall q_{o\iota}\ [\sim q\ [g\ x] \vee q\ x] \\
&\wedge \forall z_\iota.\exists q\ [q\ [g\ z] \wedge \sim q\ z] \vee \forall q.\sim q\ z \vee q\ x] \\
&\wedge \forall y_\iota\ \exists q.q\ [f\ y] \wedge \sim q\ y
\end{aligned}$$

It then skolemizes the formula (using capital letters as skolem constants) and displays the formula (with type symbols deleted) in a vpform which is shown in Figure 5-3.

Figure 5-3: Vpform for THM15B

```
|∀p^2                                                                |
| |            |          LEAF14              |              | |
| |LEAF11      |      p^2 [J^0 p^2]           |    LEAF10    | |
| |~p^2 F^0 v  |                              | v  p^2 G^0   | |
| |            |          LEAF15              |              | |
| |            | ~p^2 [λx^1 F^0 .J^0 p^2 x^1] |              | |
|                                                                    |
|            ∀q^0                                                    |
|             | LEAF21         LEAF22 |                                |
|             |~q^0 [G^0 X^0] v  q^0 X^0 |                             |
|                                                                    |
|     ∀z^0                                                           |
|       | |  LEAF28          |                          |             |
|       | |Q^0 z^0 [G^0 z^0] |   ∀q^1                   |             |
|       | |                  | v  |LEAF32     LEAF33 |  |             |
|       | |  LEAF29          |    |~q^1 z^0 v q^1 X^0 |  |             |
|       | | ~Q^0 z^0 z^0     |                          |             |
|                                                                    |
|               ∀y^0                                                 |
|                |  LEAF37          |                                |
|                |Q^1 y^0 [F^0 y^0] |                                |
|                |                  |                                |
|                |  LEAF38          |                                |
|                | ~Q^1 y^0 y^0     |                                |
```

It then starts the search for an expansion proof. It considers, in turn, the following substitutions for the predicate variable $p_{o(\iota)}$ (shown as p^2 in Figure 5-3) which was introduced in the definition of ITERATE+:

oset-0: None.
oset-1: $\lambda w^2_\iota.p^1_{o(\iota)}\ w^2 \wedge p^2_{o(\iota)}\ w^2$

Figure 5-4: Vpform for expanded form of THM15B

```
|∀p^11 p^12                                                                              |
| |                  |∀w^17                          |                                     | |
| |                  | |LEAF155   LEAF156 | |                                            | |
| | | LEAF149 |      | |p^11 J w^17 v  p^12 J w^17| |                                    | |
| | | ~p^11 F^0 W^a| |                               |      ∀w^18                          | |
| | |         | v    |        LEAF158                | v  |LEAF161          LEAF162 |     | |
| | | LEAF150 |      | ~p^11 [λx^1 F^0 .J x^1] W^b  |    |p^11 G^0 w^18  v  p^12 G^0 w^18 | | |
| | | ~p^12 F^0 W^a| |                               |                                     | |
| |                  |       LEAF159                 |                                     | |
| |                  | ~p^12 [λx^1 F^0 .J x^1] W^b  |                                     | |
|                                                                                          |
|                      ∀q^0                                                                |
|                       |  LEAF21        LEAF22 |                                         |
|                       |~q^0 [G^0 X^0]  v  q^0 X^0 |                                      |
|                                                                                          |
|            ∀z^0                                                                          |
|             | |   LEAF28        |                           |                            |
|             | |Q^0 z^0 [G^0 z^0]|    ∀q^1                   |                            |
|             | |                 | v  |LEAF32    LEAF33 |    |                            |
|             | |   LEAF29        |    |~q^1 z^0 v q^1 X^0 | |                            |
|             | |  ~Q^0 z^0 z^0   |                           |                            |
|                                                                                          |
|                        ∀y^0                                                              |
|                        |   LEAF37       |                                                |
|                        |Q^1 y^0 [F^0 y^0] |                                              |
|                        |                |                                                |
|                        |   LEAF38       |                                                |
|                        |  ~Q^1 y^0 y^0  |                                                |
where
J  is [J^0 .λw^8 ∀w^7 .p^11 w^8 w^7 v p^12 w^8 w^7]
W^a is [W^2 .λw^8 ∀w^7 .p^11 w^8 w^7 v p^12 w^8 w^7]
W^b is [W^3 .λw^8 ∀w^7 .p^11 w^8 w^7 v p^12 w^8 w^7]
```

---

oset-2: $\lambda w^3_\iota . p^3_{o(\iota)}\ w^3 \lor p^4_{o(\iota)}\ w^3$
oset-3: $\lambda w^4_\iota\ \exists w^3_\alpha\ p^5_{o(\alpha)(\iota)}\ w^4\ w^3$
oset-4: $\lambda w^5_\iota\ \forall w^4_\alpha\ p^6_{o(\alpha)(\iota)}\ w^5\ w^4$
oset-5: $\lambda w^6_\iota\ \exists w^5_\alpha . p^7_{o(\alpha)(\iota)}\ w^6\ w^5 \lor p^8_{o(\alpha)(\iota)}\ w^6\ w^5$
oset-6: $\lambda w^7_\iota\ \exists w^6_\alpha . p^9_{o(\alpha)(\iota)}\ w^7\ w^6 \land p^{10}_{o(\alpha)(\iota)}\ w^7\ w^6$

It spends about 19 minutes exploring each of these expansion options. Finally it comes to

oset-7: $\lambda w^8_\iota\ \forall w^7_\alpha . p^{11}_{o(\alpha)(\iota)}\ w^8\ w^7 \lor p^{12}_{o(\alpha)(\iota)}\ w^8\ w^7$

It applies the substitution and preprocesses to obtain a jform which is displayed in Figure 5-4.

It searches on this jform for an acceptable mating, and finds the following:

```
(LEAF28 . LEAF162)  (LEAF38 . LEAF33)  (LEAF37 . LEAF32)
```

```
(LEAF29 . LEAF22)   (LEAF21 . LEAF161)   (LEAF159 . LEAF156)
(LEAF158 . LEAF155)   (LEAF150 . LEAF149)
```

The substitution associated with the mating is:

$q^0_{o\iota}$ -> $\lambda w^{122}_{\iota}\ Q^0_{o\iota\iota}\ [F^0_{\iota\iota}\ X^0_{\iota}].F^0\ w^{122}$

z^0_{ι} -> $F^0_{\iota\iota}\ X^0_{\iota}$

$p^{11}_{o(o\iota)(\iota\iota)}$ -> $\lambda w^{123}_{\iota\iota}\ \lambda w^{124}_{o\iota}\ w^{124}.F^0_{\iota\iota}.w^{123}\ X^0_{\iota}$

$w^{17}_{o\iota}$ -> $\lambda w^{125}_{\iota}\ W^3_{o\iota(o(\iota\iota))}\ [\lambda w^8_{\iota\iota}\ \Pi_{o(o(o\iota))}.\lambda w^7_{o\iota}.w^7\ [F^0_{\iota\iota}.w^8\ X^0_{\iota}]$
$\lor \sim w^7.w^8.F^0\ X^0].F^0\ w^{125}$

$w^{18}_{o\iota}$ -> $\lambda w^{122}_{\iota}\ Q^0_{o\iota\iota}\ [F^0_{\iota\iota}\ X^0_{\iota}]\ w^{122}$

$p^{12}_{o(o\iota)(\iota\iota)}$ -> $\lambda w^{120}_{\iota\iota}\ \lambda w^{121}_{o\iota}.\sim w^{121}.w^{120}.F^0_{\iota\iota}\ X^0_{\iota}$

$q^1_{o\iota}$ -> $\lambda w^{122}_{\iota}\ Q^1_{o\iota\iota}\ X^0_{\iota}\ w^{122}$

y^0_{ι} -> X^0_{ι}

Let us examine the computation of the important compound substitution for the predicate variable $p_{o(\iota\iota)}$. The substitution term from oset-7 is:

$\lambda w^8_{\iota\iota}\ \forall w^7_{o\iota}.p^{11}_{o(o\iota)(\iota\iota)}\ w^8\ w^7 \lor p^{12}_{o(o\iota)(\iota\iota)}\ w^8\ w^7$

Applying the substitutions for $p^{11}_{o(o\iota)(\iota\iota)}$ and $p^{12}_{o(o\iota)(\iota\iota)}$ produces:

$\lambda w^8_{\iota\iota}\ \forall w^7_{o\iota}.[\lambda w^{123}_{\iota\iota}\ \lambda w^{124}_{o\iota}\ w^{124}.F^0_{\iota\iota}.w^{123}\ X^0_{\iota}]\ w^8\ w^7$
$\lor\ [\lambda w^{120}_{\iota\iota}\ \lambda w^{121}_{o\iota}.\sim w^{121}.w^{120}.F^0\ X^0]\ w^8\ w^7$

λ-normalizing transforms this to:

$\lambda w^8_{\iota\iota}\ \forall w^7_{o\iota}.w^7\ [F^0_{\iota\iota}.w^8\ X^0_{\iota}] \lor \sim w^7.w^8.F^0\ X^0$

TPS makes alphabetic changes of the variables to convert this to:

$\lambda j_{\iota\iota}\ \forall P_{o\iota}.P\ [F^0_{\iota\iota}.j\ X^0_{\iota}] \lor \sim P.j.F^0\ X^0$

TPS then constructs an expansion tree from this mating, merges it, and constructs the natural deduction proof in Figures 5-1 and 5-2, guided by the information in the expansion tree. To see how this works, let us look at two stages in this process.

By applying rules of inference in both forward and backward directions in a rather natural way, and using the substitutions for z^0_{ι} and y^0_{ι}, TPS constructs the partial proof displayed in Figure 5-5. At this stage the proof contains only lines (1)-(9) and (30)-(35). TPS is planning to prove (30), and it knows that only (5), (7), and (9) need be actively used to do this. The other lines are *inactive*, and will not be used again in the process of constructing the proof. This *status information* is represented simply as (30 9 7 5); the first entry is the number of the line to be proved, and the other entries are numbers of lines which may now be used to prove that line. In the figure we display the active lines and only the numbers of the inactive lines which are now present in the proof.

TPS next derives (10) by applying universal instantiation to (5), using the substitution for $p_{o(\iota\iota)}$

Figure 5-5: Early stage in construction of proof of THM15B

```
(1-4)
(5)   1,2    ⊢ ∀p_{o(ιι)} .p f_{ιι} ∧ ∀j_{ιι} [p j ⊃ p .λx_ι f .j x] ⊃ p g_{ιι}
                                                          EquivWffs: 3
(6)
(7)   1,2,6  ⊢ g_{ιι} x_ι = x                                  RuleP: 6
(8)
(9)   1,2,6  ⊢ g_{ιι} [f_{ιι} x_ι] = f x ⊃ f x = x     UI: [f_{ιι} x_ι] 8
              ...
(30)  1,2,6  ⊢ f_{ιι} x_ι = x                                    PLAN21
(31-35)
```

discussed above. This makes (5) inactive, so the status information is now (30 10 9 7).

With a few more inferences the proof reaches the form in Figure 5-6. Now the status information is (28 11 9)(30 29 11 9), which means that both (28) and (30) are to be proven.

Figure 5-6: Later stage in construction of proof of THM15B

```
(1-8)
(9)   1,2,6  ⊢ g_{ιι} [f_{ιι} x_ι] = f x ⊃ f x = x     UI: [f_{ιι} x_ι] 8
(10)
(11)  1,2    ⊢      ∀P_{oι} [P [f_{ιι} .f x_ι] ∨ ~P .f .f x]
                ∧ ∀j_{ιι} [  ∀P [P [f .j x] ∨ ~P .j .f x]
                           ⊃ ∀P .P [f .f .j x] ∨ ~P .f .j .f x]
              ⊃ ∀P .P [f .g_{ιι} x] ∨ ~P .g .f x           Lambda: 10
              ...
(28)  1,2    ⊢ g_{ιι} [f_{ιι} x_ι] = f .g x                    PLAN28
(29)  1,2,6  ⊢ g_{ιι} [f_{ιι} x_ι] = f x                 Subst=: 28 7
(30)  1,2,6  ⊢ f_{ιι} x_ι = x                                    PLAN21
(31-35)
```

It is hard to explain exactly how TPS decided to infer (29) from (7) and (28) without a detailed discussion of the tactics for dealing with equality which were invoked by setting the flag REMOVE-LEIBNIZ to T for this proof. Suffice it to say that (7) is descended from LEAF21 and LEAF22 in Figure 5-4, (28) is descended from LEAF161, and LEAF21 is mated to LEAF161 in the expansion proof. (The process is easier to understand when REMOVE-LEIBNIZ is NIL, but the proof thus obtained is not so elegant.)

Similarly, the antecedent of (9) is descended from LEAF28 and LEAF 29, and (29) is descended from LEAF22, so the mating between LEAF29 and LEAF22 guides the derivation of (30) by Modus Ponens (MP) from (9) and (29).

Since the consequent of (11) is essentially the assertion in (28) (modulo the Leibniz definition of equality and the symmetry of disjunction), it can be seen that the same general methods suffice to complete the construction of the proof.

6. Theorems Proved Automatically

While TPS is still in a rudimentary state as a system for automatically proving serious theorems of type theory, it is a well developed platform for experimenting with these theorems and developing ideas about the basic issues involved. In this section we discuss some examples of theorems which TPS has proved completely automatically.

Naturally, TPS can be used to prove theorems of first-order logic, but we focus mainly on examples from higher-order logic. For ease of reference, we list the theorems in the order of their labels; these simply reflect the way examples have been put into our library over the years. Theorems (such as X2129) whose names start with an "X" are exercises in [7] (or will be exercises in the next edition), and are available in TPS and ETPS whether or not one has any library files.

When TPS proves a theorem in automatic mode, it records the time used to do a number of things, including searching for an acceptable mating (*search*), merging the expansion proof (*merge*), translating the expansion proof into a natural deduction proof (*translate*), and printing the proof on the screen (*print*). It also records the total time used to do all these things and produce a natural deduction proof of the theorem (*total*). For each example below we give the internal runtime minus garbage-collect time used by TPS for some or all of these processes while running on a Hewlett Packard Apollo 9000 model 735 workstation equipped with 208 megabytes of RAM. Times are in seconds (*secs*), minutes (*mins*), or hours (*hrs*), as seems most convenient. These numbers are useful only for their approximate magnitudes; they are quite dependent on how various flags are set, and in many cases probably do not represent optimal settings of the flags. It should also be noted that the time to produce output on the screen is not negligible. In one run for THM47 which is reported below, the total runtime was 11.31 seconds. However, when we ran this again in a mode which minimized output to the screen, the total runtime was 9.50 seconds.

It will be noted that in many cases the process of translating an expansion proof into a natural deduction takes a surprisingly large amount of time, even though it involves no deep search or backtracking. This is because the conditions checked on expansion proofs in some steps of the translation are computationally expensive in order to arrive at the most natural proof possible with the current tactics. Furthermore, no attempt has been made to optimize this part of the program.

Definitions

The definitions below, which are used in various theorems we shall discuss, are built into TPS or stored in the TPS library, and the user can easily add more definitions to the library. The way TPS handles definitions during the search process is determined by the settings of certain flags, such as REWRITE-DEFNS and REWRITE-EQUALITIES. When these flags are set to T, TPS simply eliminates these definitions from the theorem while preparing the expansion tree for the search process. In some cases, this expands the search space in a very undesirable way, and more sophisticated ways of deciding when to instantiate definitions in the search process are clearly needed. (Discussions of this issue may be found in [12], [15] (where "peeking" is discussed), and [51].) Once TPS finds an expansion proof, the translation tactics cause the definitions to be handled rather naturally in the final natural deduction proof.

We remark that a wff of the form $[\lambda \mathbf{x}_\alpha\ \mathbf{B}]$, where $\mathbf{B}$ is a statement about $\mathbf{x}_\alpha$, denotes the set $\{\mathbf{x}_\alpha \mid \mathbf{B}\}$. Also, $[\mathbf{P}_{o\alpha}\ \mathbf{x}_\alpha]$ means $[\mathbf{x}_\alpha \in \mathbf{P}_{o\alpha}]$. Binary operators are often written in infix position.

ε (set membership): $\lambda x_{\alpha}\, \lambda p_{o\alpha}\, p\, x$

$\subseteq$ (subset): $[\lambda p_{o\alpha}\, \lambda r_{o\alpha}\, \forall x_{\alpha}\, .p\, x \supset r\, x]$

$\cup$ (union): $[\lambda p_{o\alpha}\, \lambda r_{o\alpha}\, \lambda z_{\alpha}\, .p\, z \vee r\, z]$

$\bigcup$ (union of a collection of sets): $[\lambda w_{o(o\alpha)}\, \lambda x_{\alpha}\, \exists s_{o\alpha}\, .w\, s \wedge s\, x]$

$\bigcap$ (intersection of a collection of sets): $[\lambda w_{o(o\alpha)}\, \lambda x_{\alpha}\, \forall s_{o\alpha}.w\, s \supset s\, x]$

$\circ$ (composition of functions): $[\lambda f_{\alpha\beta}\, \lambda g_{\beta\chi}\, \lambda x_{\chi}\, f\, .g\, x]$

INJECTIVE: $[\lambda f_{\alpha\beta}\, \forall x_{\beta}\, \forall y_{\beta}\, .f\, x = f\, y \supset x = y]$

# (image): $[\lambda f_{\alpha\beta}\, \lambda x_{o\beta}\, \lambda z_{\alpha}\, \exists t_{\beta}.x\, t \wedge z = f\, t]$

$[\#\, f_{\alpha\beta}\, x_{o\beta}]$ is the image of the set $x_{o\beta}$ under the function $f_{\alpha\beta}$.

U (unit set): $[\lambda x_{\alpha}\, \lambda y_{\alpha}\, .x = y]$

$[U\, x]$ is customarily written as $\{x\}$.

IND: $\forall p_{o\iota}\, .p\, 0_{\iota} \wedge \forall x_{\iota}\, [p\, x \supset p\, .S_{\iota\iota}\, x] \supset \forall x\, p\, x$

IND expresses a simple induction axiom for the natural numbers. 0_{ι} is zero, and $S_{\iota\iota}$ is the successor function.

ITERATE+: $[\lambda f_{\alpha\alpha}\, \lambda g_{\alpha\alpha}\, \forall p_{o(\alpha\alpha)}\, .p\, f \wedge \forall j_{\alpha\alpha}\, [p\, j \supset p\, .f \circ j] \supset p\, g]$

[ITERATE+ f g] means that g is a composition of one or more copies of f. Note how easy it is to express this inductive definition in type theory.

ITERATE: $[\lambda f_{\alpha\alpha}\, \lambda g_{\alpha\alpha}\, \forall p_{o(\alpha\alpha)}.p\, [\lambda u_{\alpha}\, u] \wedge \forall j_{\alpha\alpha}\, [p\, j \supset p.f \circ j] \supset p\, g]$

[ITERATE f g] means that g is a composition of zero or more copies of f.

HOMOM2: $[\lambda h_{\alpha\beta}\, \lambda f_{\beta\beta\beta}\, \lambda g_{\alpha\alpha\alpha}\, \forall x_{\beta}\, \forall y_{\beta}\, .h\, [f\, x\, y] = g\, [h\, x]\, .h\, y]$

[HOMOM2 h f g] means that h is a homomorphism from objects of type β to objects of type α, where f and g are binary operators on the types β and α, respectively.

MAPS: $[\lambda h_{\alpha\beta}\, \lambda u_{o\beta}\, \lambda v_{o\alpha}\, \forall x_{\beta}.u\, x \supset v.h\, x]$

[MAPS h u v] means that the function h maps the set u into the set v.

-CLOSED: $[\lambda h_{\alpha\alpha}\, \lambda u_{o\alpha}\, \mathrm{MAPS}_{o(o\alpha)(o\alpha)(\alpha\alpha)}\, h\, u\, u]$

[h -CLOSED u] means that the set u is closed under the function h.

HOM: $\lambda h_{\alpha\beta}\, \lambda r_{o\beta}\, \lambda f_{\beta\beta}\, \lambda s_{o\alpha}\, \lambda g_{\alpha\alpha}.f\ \text{-CLOSED}\ r \wedge g\ \text{-CLOSED}\ s \wedge \mathrm{MAPS}_{o(o\alpha)(o\beta)(\alpha\beta)}\, h\, r\, s \wedge \forall x_{\beta}.r\, x \supset h\, [f\, x] = g.h\, x$

[HOM h r f s g] means that h is a homomorphism from <r,f> to <s,g>, where r and s are sets, f is a unary operator on r, and g is a unary operator on s.

Theorems

THM15B: $\forall f_{\iota\iota}.\exists g_{\iota\iota}\, [\text{ITERATE+}\, f\, g \wedge \exists x_{\iota}.g\, x = x \wedge \forall z_{\iota}.g\, z = z \supset z = x] \supset \exists y_{\iota}.f\, y = y$

(search: 2.5 hrs total: 2.5 hrs)

This theorem was discussed in section 5.

THM30: $R_{o\alpha} \subseteq S_{o\alpha} \equiv \forall F_{\alpha\alpha}.\#\, F\, R \subseteq \#\, F\, S$ (search: 0.56 secs translate: 1.19 secs total: 2.52 secs)

THM47: $\forall X_{\iota}\, \forall Y_{\iota}\, .\forall Q_{o\iota}\, [Q\, X \supset Q\, Y] \equiv \forall R_{o\iota\iota}\, .\forall Z_{\iota}\, R\, Z\, Z \supset R\, X\, Y$

Run with MATING-VERBOSE MAX and UNIFY-VERBOSE MAX:

(search: 9.93 secs total: 11.31 secs)

Run with MATING-VERBOSE SILENT and UNIFY-VERBOSE SILENT:
(search: 8.02 secs total: 9.50 secs)
THM47 shows the equivalence of two ways of defining equality in type theory: the Leibniz definition, and the intersection of all reflexive relations.

THM48: $\forall F_{\alpha\beta}\ \forall G_{\beta\chi}$.INJECTIVE F $\wedge$ INJECTIVE G $\supset$ INJECTIVE.F $\circ$ G
Trial with REWRITE-EQUALITIES set to T:
(search: 89.74 secs total: 91.43 secs)
Trial with REWRITE-EQUALITIES set to NIL:
(search: 0.04 secs total: 0.81 secs)
THM48 asserts that the composition of injective functions is injective. The definition of equality which is contained in the definition of INJECTIVE is actually not needed in order to prove this theorem, and the time required to prove the theorem is dramatically affected by whether the equalities are instantiated or not.

THM67: $\forall S_{o\alpha}\ \forall T_{o\alpha}\ [S \subseteq T \supset F_{o\alpha(o\alpha)}\ T \subseteq F\ S] \wedge [\forall S\ [S \subseteq F\ [G_{o\alpha(o\alpha)}\ S]] \wedge \forall S\ [S \subseteq G\ [FS]]] \supset \forall S$
$\forall x_\alpha\ [F\ [G\ [F\ S]]\ x \equiv FS\ x]$ (search: 11.79 secs total: 12.75 secs)

Next we have two examples which were discussed in [8] as examples of theorems which require instantiations for set-variables which cannot be obtained by unification from literals in the theorem.

THM104: $\forall X_\alpha\ \forall Z_\alpha . U\, X = U\, Z \supset X = Z$ (search: 9.67 secs total: 10.55 secs)
The proof referred to here was obtained using the Leibniz definition of equality, and uses a projection as an expansion term. However, if we change the value of the flag REWRITE-EQUAL-EXT so that TPS uses the extensional definition of equality between sets, no expansion option is needed, and the times for the proof are: (search: 0.14 secs total: 1.22 secs)

THM112: $\forall P_{o\iota}\ \exists M_{o(\iota\iota)}\ \forall G_{\iota\iota}\ \forall H_{\iota\iota} .\ M\, G \wedge M\, H \supset M\, [G \circ H] \wedge \forall Y_\iota .P\, Y \supset P\, .G\, Y$
(search: 6.13 secs total: 6.13 secs)
THM112 asserts that for any set P, there is a set M of functions mapping P into P which is closed under composition. TPS quickly finds a trivial proof where the set M is $[\lambda w_{\iota\iota}.\sim h_o \wedge h]$, i.e., the empty set of functions. To make the problem slightly less trivial, we excluded this solution in the statement of THM112A below.

THM112A: $\forall P_{o\iota}\ \exists M_{o(\iota\iota)}.M\ [\lambda x_\iota\ x] \wedge \forall G_{\iota\iota}\ \forall H_{\iota\iota}.M\, G \wedge M\, H \supset M\, [G \circ H] \wedge \forall Y_\iota.P\, Y \supset P.G\, Y$
(search: 4.4 mins total: 4.4 mins)
For the proof of THM112A, TPS finds that it suffices to let M be $[\lambda f_{\iota\iota}\ \forall u_\iota.P_{o\iota}\ [f\ u] \vee \sim P\ u]$. This can be rewritten as $[\lambda f_{\iota\iota}\ \forall u_\iota.P_{o\iota}\ u \supset P.f\ u]$, and denotes the set of functions which map P into itself.

THM117C: $\forall x_{o\iota}\ \forall z_\iota\ [z\ \varepsilon\ x \supset \exists y_\iota.y\ \varepsilon\ x \wedge \forall w_\iota.R_{o\iota\iota}\ y\ w \supset \sim w\ \varepsilon\ x] \wedge \forall x1_\iota\ [\forall y1_\iota\ [y1\ \varepsilon\ s_{o\iota} \wedge R\ x1\ y1 \supset$
$P_{o\iota}\ y1] \supset P\ x1] \supset \forall x2_\iota.x2\ \varepsilon\ s \supset P\ x2$ (search: 0.19 secs total: 1.63 secs)
This is the TRANSFINITE INDUCTION theorem of [11] (page 396) expressed in the language of type theory. Think of Ryw as saying that y > w. The theorem asserts that if R is a well-founded relation and P is an inductive property over R restricted to the set s, then everything in s has property P.

THM129: $\text{IND} \wedge \forall x_\iota\ +_{o\iota\iota}\ 0_\iota\ x\ x \wedge \forall x\ \forall y_\iota\ \forall z_\iota\ [+\ y\ x\ z \supset +\ [S_{\iota\iota}\ y]\ x.S\ z] \supset \forall y\ \forall x\ \exists z +\ y\ x\ z$
(search: 0.57 secs total: 2.33 secs)

THM130: $\mathrm{IND} \wedge r_{o\iota\iota}\, 0_\iota\, 0 \wedge \forall x_\iota\, \forall y_\iota\, [r\, x\, y \supset r\, [S_{\iota\iota}\, x]\, .S\, y] \supset \forall x\, \exists y\, r\, x\, y$
(search: 0.51 secs total: 1.18 secs)
This is a theorem in which the conclusion is weaker than the statement which must be proved by induction. From the hypotheses Tps proves $\forall x_\iota\, r_{o\iota\iota}\, x\, x$ by induction, and from this derives the desired conclusion $\forall x_\iota\, \exists y_\iota\, r_{o\iota\iota}\, x\, y$. No special mechanism for deciding what to prove by induction is built into Tps; it falls naturally out of a purely logical analysis of the structure of THM130.

THM131: $\forall h1_{\beta\gamma}\, \forall h2_{\alpha\beta}\, \forall s1_{o\gamma}\, \forall f1_{\gamma\gamma}\, \forall s2_{o\beta}\, \forall f2_{\beta\beta}\, \forall s3_{o\alpha}\, \forall f3_{\alpha\alpha}.\mathrm{HOM}\ h1\ s1\ f1\ s2\ f2 \wedge \mathrm{HOM}\ h2\ s2\ f2\ s3\ f3 \supset \mathrm{HOM}\ [h2 \circ h1]\ s1\ f1\ s3\ f3$
(search: 30.5 mins merging 6.2 mins translate: 3.99 secs total: 36.8 mins)
This example, which asserts that the composition of homomorphisms is a homomorphism, was suggested in [18], though the formulation of the theorem given there in terms of the primitives of axiomatic set theory makes it much harder to prove. It may be enlightening to compare this with THM133 below, which is much easier for Tps since the closure of the sets under the appropriate functions is dealt with implicitly through the use of types in THM133.

THM133: $\forall h1_{\beta\gamma}\ \forall h2_{\alpha\beta}\ \forall f1_{\gamma\gamma\gamma}\ \forall f2_{\beta\beta\beta}\ \forall f3_{\alpha\alpha\alpha}\ .\ \mathrm{HOMOM2}\ h1\ f1\ f2 \wedge \mathrm{HOMOM2}\ h2\ f2\ f3 \supset \mathrm{HOMOM2}\ [h2 \circ h1]\ f1\ f3$ (search: 3.38 secs total: 5.15 secs)

THM134: $\forall z_\iota\, \forall g_{\iota\iota}\, .\mathrm{ITERATE+}\ [\lambda x_\iota\, z]\ g \supset \forall x\, .g\, x = z$ (search: 0.05 secs total: 1.02 secs)
THM134 can be paraphrased as saying that the only positive iterate of a constant function is that function.

THM135: $\forall f_{\alpha\alpha}\, \forall g1_{\alpha\alpha}\, \forall g2_{\alpha\alpha}.\mathrm{ITERATE}\ f\ g1 \wedge \mathrm{ITERATE}\ f\ g2 \supset \mathrm{ITERATE}\ f.g1 \circ g2$
(search: 3.4 mins total: 3.5 mins)
This theorem asserts that the composition of iterates of a function is an iterate of that function.

In the next two theorems [DOUBLE u v] means that 2u = v, and [HALF u v] means that the greatest integer in u/2 is v.

THM300A: $\forall u_\iota\, \forall v_\iota\, [\mathrm{HALF}_{o\iota\iota}\, u\, v \equiv \forall Q_{o\iota\iota}.Q\, 0_\iota\, 0 \wedge Q\, [S_{\iota\iota}\, 0]\, 0 \wedge \forall x_\iota\, \forall y_\iota\, [Q\, x\, y \supset Q\, [S.S\, x].S\, y] \supset Q\, u\, v] \wedge \mathrm{DOUBLE}_{o\iota\iota}\, 0\, 0 \wedge \forall x\, \forall y\, [\mathrm{DOUBLE}\, x\, y \supset \mathrm{DOUBLE}\, [S\, x].S.S\, y] \supset \forall u\, \forall v.\mathrm{HALF}\, u\, v \supset \mathrm{DOUBLE}\, v\, u \vee \mathrm{DOUBLE}\, [S\, v].S\, u$
(search: 83.56 secs total: 88.13 secs)

THM301A: $\forall u_\iota\, \forall v_\iota\, [\mathrm{DOUBLE}_{o\iota\iota}\, u\, v \equiv \forall Q_{o\iota\iota}.Q\, 0_\iota\, 0 \wedge \forall x_\iota\, \forall y_\iota\, [Q\, x\, y \supset Q\, [S_{\iota\iota}\, x].S.S\, y] \supset Q\, u\, v] \wedge \mathrm{HALF}_{o\iota\iota}\, 0\, 0 \wedge \mathrm{HALF}\, [S\, 0]\, 0 \wedge \forall x\, \forall y\, [\mathrm{HALF}\, x\, y \supset \mathrm{HALF}\, [S.S\, x].S\, y] \supset \forall u\, \forall v.\mathrm{DOUBLE}\, u\, v \supset \mathrm{HALF}\, v\, u$ (search: 56.90 secs total: 62.79 secs)

THM303: $\mathrm{EVEN}_{o\iota}\, 0_\iota \wedge \forall n_\iota\, [\mathrm{EVEN}\, n \supset \mathrm{EVEN}.S_{\iota\iota}.S\, n] \wedge [\mathrm{ODD}_{o\iota}\, [S\, 0] \wedge \forall n.\mathrm{ODD}\, n \supset \mathrm{ODD}.S.S\, n] \wedge \mathrm{IND} \wedge \forall n\, [\mathrm{NUMBER}_{o\iota}\, n \equiv \mathrm{EVEN}\, n \vee \mathrm{ODD}\, n] \supset \forall n\, \mathrm{NUMBER}\, n$
(search: 33.8 mins total: 33.9 mins)
After assuming the antecedent, Tps proves $\forall x_\iota\, [\mathrm{NUMBER}_{o\iota}\, x \wedge \mathrm{NUMBER}\, .S_{\iota\iota}\, x]$ by induction (using IND), and from this derives $\forall n_\iota\, \mathrm{NUMBER}_{o\iota}\, n$. Note that a direct inductive proof of $\forall n_\iota\, \mathrm{NUMBER}_{o\iota}\, n$ does not work.

BLEDSOE-FENG-SV-I1: $\forall A_{o\iota}\, [A\, O_\iota \wedge \forall x_\iota\, [A\, x \supset A.1+_{\iota\iota}\, x] \supset A\, n_\iota] \wedge P_{o\iota}\, O \wedge \forall x\, [P\, x \supset P.1+\, x] \supset P\, n$ (search: 0.14 secs total: 0.61 secs)

This is Example I1 from [17].

BLEDSOE-FENG-SV-I2: $\forall A_{o\iota\iota}\, [A\, O_\iota\, O \land \forall x_\iota\, \forall y_\iota\, [A\, x\, y \supset A\, [s_{\iota\iota}\, x].s\, y] \supset A\, n_\iota\, m_\iota] \land P_{o\iota}\, n \supset P\, m$
(search: 8.15 secs total: 9.91 secs)

This is Example I2 from [17].

X2115: $\forall x_\iota\, [\exists y_\iota\, P_{o\iota\iota}\, x\, y \supset \forall z_\iota\, P\, z\, z] \land \forall u_\iota\, \exists v_\iota\, [P\, u\, v \lor M_{o\iota}\, u \land Q_{o\iota}.f_{\iota\iota\iota}\, u\, v] \land \forall w_\iota\, [Q\, w \supset {\sim}M.g_{\iota\iota}\, w]$
$\supset \forall u\, \exists v.P\, [g\, u]\, v \land P\, u\, u$
(search: 0.12 secs merge 0.72 secs translate 2.57 secs total: 4.73 secs)

X2116: $\forall x_\iota\, \exists y_\iota\, [P_{o\iota}\, x \supset R_{o\iota\iota}\, x\, [g_{\iota\iota}.h_{\iota\iota}\, y] \land P\, y] \land \forall w_\iota\, [P\, w \supset P\, [g\, w] \land P.h\, w] \supset \forall x.P\, x \supset \exists y.R\, x\, y \land$
$P\, y$ (search: 0.42 secs total: 0.93 secs)

X2129: $\exists x\, \forall y\, [P\, x \equiv P\, y] \equiv [\exists x\, Q\, x \equiv \forall y\, P\, y] \equiv .\exists x\, \forall y\, [Q\, x \equiv Q\, y] \equiv .\exists x\, P\, x \equiv \forall y\, Q\, y$
(search: 0.14 secs merge: 44.05 secs translate: 70.87 secs print: 5.67 secs total: 122.48 secs)
This was presented as a challenge problem by Andrews at the Fourth Workshop on Automated Deduction in 1979. Other researchers (see references cited in [32] and [47]) have found ways to deal with this problem, often by making reductions of it in the preprocessing stage. TPS found a refutation for this problem using path-focused duplication. The natural deduction proof is 584 lines long.

X5200: $x \cup y = \bigcup .\lambda v\, .v = x \lor v = y$ (search: 13.45 secs total: 16.62 secs)

X5205: $\#\, f_{\alpha\beta}\, [\bigcap w_{o(o\beta)}] \subseteq \bigcap .\#\, [\#\, f]\, w$ (search: 4.44 mins total: 6.31 mins)

X5304: ${\sim}\exists g_{o\alpha\alpha}\, \forall f_{o\alpha}\, \exists j_\alpha\, .g\, j = f$ (search: 0.09 secs total: 0.43 secs)
This is the Simple Cantor Theorem for Sets, which TPS1 could prove [6]. It is stated here for comparison with X5305 below, which is harder to prove.

X5305: $\forall s_{o\alpha}\, .{\sim}\exists g_{o\alpha\alpha}\, \forall f_{o\alpha}\, .f \subseteq s \supset \exists j_\alpha\, .s\, j \land g\, j = f$ (search: 0.37 secs total: 1.42 secs)
We call this the General Cantor Theorem for Sets. It says that there is no mapping g from a set s onto its power set.

X5308: $\exists j_{\beta(o\beta)}\, \forall p_{o\beta}\, [\exists x_\beta\, p\, x \supset p.j\, p] \supset .\forall x_\alpha\, \exists y_\beta\, r_{o\beta\alpha}\, x\, y \equiv \exists f_{\beta\alpha}\, \forall x\, r\, x.f\, x$
(search: 0.31 secs total: 1.21 secs)
The Axiom of Choice (for type β) is $[\exists j_{\beta(o\beta)}\, \forall p_{o\beta}.\exists x_\beta\, p\, x \supset p.j\, p]$; it asserts that there is a choice function $j_{\beta(o\beta)}$ which chooses an element $j_{\beta(o\beta)}p_{o\beta}$ from every non-empty set $p_{o\beta}$. X5308 shows a consequence of this axiom.

X5310: $\forall r_{o\beta(o\beta)}\, [\forall x_{o\beta}\, \exists y_\beta\, r\, x\, y \supset \exists f_{\beta(o\beta)}\, \forall x\, r\, x.f\, x] \supset \exists j_{\beta(o\beta)}\, \forall p_{o\beta}.\exists x_\beta\, p\, x \supset p.j\, p$
(search: 21.7 mins total: 21.8 mins)
X5310 implies the converse of X5308 (suitably generalized) when α is $(o\beta)$. The proof of this theorem was discussed in section 4.

7. Conclusion

Much additional development and study of TPS is still needed. Nevertheless, it already provides a rich environment for exploring the complexities of theorem proving in higher-order logic. With the aid of TPS, one can think in a concrete way about these problems. TPS is also a convenient tool for proving theorems interactively or semi-automatically.

8. References

1. Peter B. Andrews, *Resolution in Type Theory*, Journal of Symbolic Logic **36** (1971), 414-432.

2. Peter B. Andrews, *Provability in Elementary Type Theory*, Zeitschrift fur Mathematische Logic und Grundlagen der Mathematik **20** (1974), 411-418.

3. Peter B. Andrews, *Refutations by Matings*, IEEE Transactions on Computers **C-25** (1976), 801-807.

4. Peter B. Andrews, "Transforming Matings into Natural Deduction Proofs," in *5th Conference on Automated Deduction*, edited by W. Bibel and R. Kowalski, Les Arcs, France, Lecture Notes in Computer Science 87, Springer-Verlag, 1980, 281-292.

5. Peter B. Andrews, *Theorem Proving via General Matings*, Journal of the ACM **28** (1981), 193-214.

6. Peter B. Andrews, Dale A. Miller, Eve Longini Cohen, Frank Pfenning, "Automating Higher-Order Logic," in *Automated Theorem Proving: After 25 Years*, edited by W. W. Bledsoe and D. W. Loveland, Contemporary Mathematics series, vol. 29, American Mathematical Society, 1984, 169-192.

7. Peter B. Andrews, *An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof*, Academic Press, 1986.

8. Peter B. Andrews, *On Connections and Higher-Order Logic*, Journal of Automated Reasoning **5** (1989), 257-291.

9. Peter B. Andrews, Sunil Issar, Dan Nesmith, Frank Pfenning, Hongwei Xi, Matthew Bishop, *TPS3 Facilities Guide for Programmers and Users*, 1994. 170+vii pp.

10. Peter B. Andrews, Sunil Issar, Dan Nesmith, Frank Pfenning, Hongwei Xi, Matthew Bishop, *TPS3 Facilities Guide for Users*, 1994. 103+v pp.

11. Sidney C. Bailin and Dave Barker-Plummer, *Z-match: An Inference Rule for Incrementally Elaborating Set Instantiations*, Journal of Automated Reasoning **11** (1993), 391-428.

12. Dave Barker-Plummer, *Gazing: An Approach to the Problem of Definition and Lemma Use*, Journal of Automated Reasoning **8** (1992), 311-344.

13. Wolfgang Bibel, *Automated Theorem Proving*, Vieweg, Braunschweig, 1987.

14. Matthew Bishop, Dan Nesmith, Frank Pfenning, Sunil Issar, Peter B. Andrews, Hongwei Xi, *TPS3 Programmer's Guide*, 1994. 107+iv pp.

15. W. W. Bledsoe and Peter Bruell, *A Man-Machine Theorem-Proving System*, Artificial Intelligence **5** (1974), 51-72.

16. W. W. Bledsoe, "A Maximal Method for Set Variables in Automatic Theorem Proving," in *Machine Intelligence 9*, edited by J. E. Hayes, Donald Michie, L. I. Mikulich, Ellis Harwood Ltd., Chichester, and John Wiley & Sons, 1979, pp. 53-100.

17. W. W. Bledsoe and Gohui Feng, *Set-Var,* Journal of Automated Reasoning **11** (1993), 293-314.

18. Robert Boyer, Ewing Lusk, William McCune, Ross Overbeek, Mark Stickel, and Lawrence Wos, *Set Theory in First-Order Logic: Clauses for Godel's Axioms,* Journal of Automated Reasoning 2 (1986), 287-327.

19. Alonzo Church, *A Formulation of the Simple Theory of Types,* Journal of Symbolic Logic 5 (1940), 56-68.

20. R. L. Constable et al., *Implementing Mathematics with the Nuprl Proof Development System,* Prentice Hall, 1986.

21. Thierry Coquand and Gerard Huet, *The Calculus of Constructions,* Information and Computation **76** (1988), 95-120.

22. William M. Farmer, Joshua D. Guttman, F. Javier Thayer, *IMPS: An Interactive Mathematical Proof System,* Journal of Automated Reasoning **11** (1993), 213-248.

23. Jean H. Gallier, Paliath Narendran, Stan Raatz and Wayne Snyder, *Theorem Proving Using Equational Matings and Rigid E-Unification,* Journal of the ACM **39** (1992), 377-429.

24. Doug Goldson and Steve Reeves, *Using Programs to Teach Logic to Computer Scientists,* Notices of the American Mathematical Society **40** (1993), 143-148.

25. Douglas Goldson, Steve Reeves and Richard Bornat, *A Review of Several Programs for the Teaching of Logic,* The Computer Journal **36** (1993), 373-386.

26. Michael J. Gordon, Arthur J. Milner, Christopher P. Wadsworth. *Edinburgh LCF,* Lecture Notes in Computer Science 78, Springer Verlag, 1979.

27. Michael J. C. Gordon, "HOL: A Proof Generating System for Higher-Order Logic," in *VLSI Specification, Verification, and Synthesis,* edited by Graham Birtwistle and P.A. Subrahmanyam, Kluwer Academic Publishers, 1988, pp. 73-128.

28. Leen Helmink and Rene Ahn, "Goal Directed Proof Construction in Type Theory," in *Logical Frameworks,* edited by Gerard Huet and Gordon Plotkin, Cambridge University Press, 1991, 120-148.

29. Gerard P. Huet, "A Mechanization of Type Theory," in *Proceedings of the Third International Joint Conference on Artificial Intelligence,* IJCAI, 1973, 139-146.

30. Gerard P. Huet, *A Unification Algorithm for Typed λ-Calculus,* Theoretical Computer Science 1 (1975), 27-57.

31. Sunil Issar, "Path-Focused Duplication: A Search Procedure for General Matings," in *AAAI-90. Proceedings of the Eighth National Conference on Artificial Intelligence,* AAAI Press/The MIT Press, 1990, 221-226.

32. Sunil Issar. *Operational Issues in Automated Theorem Proving Using Matings,* Ph.D. Thesis, Carnegie Mellon University, 1991. 147 pp.

33. Sunil Issar, Peter B. Andrews, Frank Pfenning, Dan Nesmith, *GRADER Manual,* 1992. 23+i pp.

34. Ignace I. Kolodner, *Fixed Points*, American Mathematical Monthly **71** (1964), 906.

35. Dale A. Miller, Eve Longini Cohen, Peter B. Andrews, "A Look at TPS," in *6th Conference on Automated Deduction*, edited by Donald W. Loveland, New York, USA, Lecture Notes in Computer Science 138, Springer-Verlag, 1982, 50-69.

36. Dale A. Miller. *Proofs in Higher-Order Logic*, Ph.D. Thesis, Carnegie Mellon University, 1983. 81 pp.

37. Dale A. Miller, "Expansion Tree Proofs and Their Conversion to Natural Deduction Proofs," in *7th International Conference on Automated Deduction*, edited by R. E. Shostak, Napa, California, USA, Lecture Notes in Computer Science 170, Springer-Verlag, 1984, 375-393.

38. Dale A. Miller, *A Compact Representation of Proofs*, Studia Logica **46** (1987), 347-370.

39. Robin Milner, *A Theory of Type Polymorphism in Programming*, Journal of Computer and System Sciences **17** (1978), 348--375.

40. Dan Nesmith, Matthew Bishop, Peter B. Andrews, Sunil Issar, Frank Pfenning, Hongwei Xi, *TPS User's Manual*, 1994. 53+ii pp.

41. Hans Jurgen Ohlbach. *A Resolution Calculus for Modal Logics*, Ph.D. Thesis, Department of Computer Science, University of Kaiserslautern, 1988.

42. Lawrence C. Paulson, *The Foundation of a Generic Theorem Prover*, Journal of Automated Reasoning **5** (1989), 363-397.

43. Frank Pfenning, "Analytic and Non-analytic Proofs," in *7th International Conference on Automated Deduction*, edited by R. E. Shostak, Napa, California, USA, Lecture Notes in Computer Science 170, Springer-Verlag, 1984, 394-413.

44. Frank Pfenning. *Proof Transformations in Higher-Order Logic*, Ph.D. Thesis, Carnegie Mellon University, 1987. 156 pp.

45. Frank Pfenning and Dan Nesmith, "Presenting Intuitive Deductions via Symmetric Simplification," in *10th International Conference on Automated Deduction*, edited by M. E. Stickel, Kaiserslautern, FRG, Lecture Notes in Artificial Intelligence 449, Springer-Verlag, 1990, 336-350.

46. Frank Pfenning, Sunil Issar, Dan Nesmith, Peter B. Andrews, Hongwei Xi, Matthew Bishop, *ETPS User's Manual*, 1994. 56+ii pp.

47. Art Quaife, *Andrews' Challenge Problem Revisited*, AAR Newsletter **15** (May 1990), 3-7.

48. J. A. Robinson, *A Machine-Oriented Logic Based on the Resolution Principle*, Journal of the ACM **12** (1965), 23-41.

49. Wayne Snyder, "Higher-Order E-Unification," in *10th International Conference on Automated Deduction*, edited by M. E. Stickel, Kaiserslautern, FRG, Lecture Notes in Artificial Intelligence 449, Springer-Verlag, 1990, 573-587.

50. Lincoln A. Wallen, *Automated Deduction in Nonclassical Logics*, MIT Press, 1990.

51. Larry Wos, *The Problem of Definition Expansion and Contraction,* Journal of Automated Reasoning **3** (1987), 433-435.