Michael Kohlhase (Ed.)

LNAI 3863

Mathematical Knowledge Management

4th International Conference, MKM 2005
Bremen, Germany, July 2005
Revised Selected Papers

Springer

Michael Kohlhase (Ed.)

Mathematical Knowledge Management

4th International Conference, MKM 2005
Bremen, Germany, July 15-17, 2005
Revised Selected Papers

Springer

Series Editors

Jaime G. Carbonell, Carnegie Mellon University, Pittsburgh, PA, USA
Jörg Siekmann, University of Saarland, Saarbrücken, Germany

Volume Editor

Michael Kohlhase
International University Bremen
School of Engineering and Science
Campus Ring 1, 28758 Bremen, Germany
E-mail: m.kohlhase@iu-bremen.de

Library of Congress Control Number: 2006920149

CR Subject Classification (1998): I.2, H.3, H.2.8, I.7.2, F.4.1, H.4, C.2.4, G.4, I.1

LNCS Sublibrary: SL 7 – Artificial Intelligence

ISSN 0302-9743
ISBN-10 3-540-31430-X Springer Berlin Heidelberg New York
ISBN-13 978-3-540-31430-1 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11618027 06/3142 5 4 3 2 1 0

Lecture Notes in Artificial Intelligence 3863

Edited by J. G. Carbonell and J. Siekmann

Subseries of Lecture Notes in Computer Science

Preface

This volume contains the proceedings of the Fourth International Conference on Mathematical Knowledge Management MKM 2005 held July 15–17, 2005 at International University Bremen, Germany. Previous conferences have been at the Research Institute for Symbolic Computation (RISC) Linz, Austria (September 2001), at Bertinoro, Italy (March 2003), and Bialowiecze, Poland (September 2004).

Mathematical knowledge management (MKM) is a field in the intersection of mathematics and computer science, providing new techniques for managing the enormous volume of mathematical knowledge available in current mathematical sources and making it available through the new developments in information technology.

The annual MKM Conference brings together mathematicians, software developers, publishing companies, math organizations, math users, and educators to exchange their views and approaches, current activities and new initiatives.

For the first time, MKM 2005 chose to have post-conference proceedings, as otherwise the submission deadline would have collided with other conferences and crimped time since MKM 2004 in September 2004. The decision also facilitated keeping the conference open to new ideas as well as keeping up the maturity of the papers necessary for inclusion into archival proceedings. With a May 15 deadline, MKM 2005 received 38 submissions. Each submission was reviewed by at least three programme committee members. The committee decided to accept 27 papers for presentation at the conference. Out of these, 26 papers were accepted for publication in the conference proceedings after re-evaluation by the Programme Committee since they included significant improvements triggered by the referee reports and the discussions at the conference.

As MKM is a small conference with a tightly knit community of authors, submissions by Programme Committee members were allowed: six submissions included committee members, but the review process was kept inaccessible to them. One submission was co-authored by the Program Chair; its review process was organized independently by Bill Farmer.

The papers in this volume cover the whole area of mathematical knowledge management. Topics range from foundations and the representational and document-structure aspects of mathematical knowledge, over process questions like authoring, migration, and consistency management by automated theorem proving to applications in eLearning and case studies.

I am grateful to Tom Hales for agreeing to give an invited talk at MKM 2005, to the Programme Committee, and the external reviewers for their excellent work and dedication to the MKM 2005 program. The work of the Programme Committee and the preparation of the proceedings were greatly simplified by Andrei Voronkov's excellent `EasyChair` system.

October 2005 Michael Kohlhase

Conference Organization

Programme Chair

Michael Kohlhase, International University Bremen, Germany

Programme Committee

Andrew A. Adams, The University of Reading, UK
Andrea Asperti, University of Bologna, Italy
Richard Baraniuk, Rice University, USA
Christoph Benzmüller, Saarland University, Germany
Olga Caprotti, University of Helsinki, Finland
Mike Dewar, NAG, Ltd., UK
William Farmer, McMaster University, Canada
Tetsuo Ida, University of Tsukuba, Japan
Fairouz Kamareddine, Heriott Watt University, UK
Robert Miner, Design Science, USA
Till Mossakowski, University of Bremen, Germany
Andrzej Trybulec, University of Bialystok, Poland
Stephen Watt, University Western Ontario, Canada

Local Organization

The local organization was carried out by the Conference Chair in collaboration with Events4, a conference organization company founded and operated by IUB students.

External Reviewers

Morten Andersen
Grzegorz Bancerek
Rebhi Baraka
Marc Buckley
Chad Brown
Paul Cairns
David Carlisle
Arjeh Cohen
Armin Fiedler
Yukiyoshi Kameyama
Artur Kornilowicz
Klaus Luettich
Christoph Lüth
Mircea Marin

Andreas Meier
Yasuhiko Minamide
Immanuel Normann
Martijn Oostdijk
Matti Pauna
Martin Pollet
Krzysztof Prazmowski
Christoph Schwarzweller
Andreas Strotmann
Diedrich Wolter
Jürgen Zimmer

Lecture Notes in Artificial Intelligence (LNAI)

Vol. 3863: M. Kohlhase (Ed.), Mathematical Knowledge Management. XI, 405 pages. 2006.

Vol. 3848: J.-F. Boulicaut, L. De Raedt, H. Mannila (Eds.), Constraint-Based Mining and Inductive Databases. X, 401 pages. 2006.

Vol. 3847: K.P. Jantke, A. Lunzer, N. Spyratos, Y. Tanaka (Eds.), Federation over the Web. X, 215 pages. 2006.

Vol. 3835: G. Sutcliffe, A. Voronkov (Eds.), Logic for Programming, Artificial Intelligence, and Reasoning. XIV, 744 pages. 2005.

Vol. 3817: M. Faundez-Zanuy, L. Janer, A. Esposito, A. Satue-Villar, J. Roure, V. Espinosa-Duro (Eds.), Nonlinear Analyses and Algorithms for Speech Processing. XII, 380 pages. 2005.

Vol. 3814: M. Maybury, O. Stock, W. Wahlster (Eds.), Intelligent Technologies for Interactive Entertainment. XV, 342 pages. 2005.

Vol. 3809: S. Zhang, R. Jarvis (Eds.), AI 2005: Advances in Artificial Intelligence. XXVII, 1344 pages. 2005.

Vol. 3808: C. Bento, A. Cardoso, G. Dias (Eds.), Progress in Artificial Intelligence. XVIII, 704 pages. 2005.

Vol. 3802: Y. Hao, J. Liu, Y.-P. Wang, Y.-m. Cheung, H. Yin, L. Jiao, J. Ma, Y.-C. Jiao (Eds.), Computational Intelligence and Security, Part II. XLII, 1166 pages. 2005.

Vol. 3801: Y. Hao, J. Liu, Y.-P. Wang, Y.-m. Cheung, H. Yin, L. Jiao, J. Ma, Y.-C. Jiao (Eds.), Computational Intelligence and Security, Part I. XLI, 1122 pages. 2005.

Vol. 3789: A. Gelbukh, Á. de Albornoz, H. Terashima-Marín (Eds.), MICAI 2005: Advances in Artificial Intelligence. XXVI, 1198 pages. 2005.

Vol. 3782: K.-D. Althoff, A. Dengel, R. Bergmann, M. Nick, T.R. Roth-Berghofer (Eds.), Professional Knowledge Management. XXIII, 739 pages. 2005.

Vol. 3763: H. Hong, D. Wang (Eds.), Automated Deduction in Geometry. X, 213 pages. 2006.

Vol. 3735: A. Hoffmann, H. Motoda, T. Scheffer (Eds.), Discovery Science. XVI, 400 pages. 2005.

Vol. 3734: S. Jain, H.U. Simon, E. Tomita (Eds.), Algorithmic Learning Theory. XII, 490 pages. 2005.

Vol. 3721: A.M. Jorge, L. Torgo, P.B. Brazdil, R. Camacho, J. Gama (Eds.), Knowledge Discovery in Databases: PKDD 2005. XXIII, 719 pages. 2005.

Vol. 3720: J. Gama, R. Camacho, P.B. Brazdil, A.M. Jorge, L. Torgo (Eds.), Machine Learning: ECML 2005. XXIII, 769 pages. 2005.

Vol. 3717: B. Gramlich (Ed.), Frontiers of Combining Systems. X, 321 pages. 2005.

Vol. 3702: B. Beckert (Ed.), Automated Reasoning with Analytic Tableaux and Related Methods. XIII, 343 pages. 2005.

Vol. 3698: U. Furbach (Ed.), KI 2005: Advances in Artificial Intelligence. XIII, 409 pages. 2005.

Vol. 3690: M. Pěchouček, P. Petta, L.Z. Varga (Eds.), Multi-Agent Systems and Applications IV. XVII, 667 pages. 2005.

Vol. 3684: R. Khosla, R.J. Howlett, L.C. Jain (Eds.), Knowledge-Based Intelligent Information and Engineering Systems, Part IV. LXXIX, 933 pages. 2005.

Vol. 3683: R. Khosla, R.J. Howlett, L.C. Jain (Eds.), Knowledge-Based Intelligent Information and Engineering Systems, Part III. LXXX, 1397 pages. 2005.

Vol. 3682: R. Khosla, R.J. Howlett, L.C. Jain (Eds.), Knowledge-Based Intelligent Information and Engineering Systems, Part II. LXXIX, 1371 pages. 2005.

Vol. 3681: R. Khosla, R.J. Howlett, L.C. Jain (Eds.), Knowledge-Based Intelligent Information and Engineering Systems, Part I. LXXX, 1319 pages. 2005.

Vol. 3673: S. Bandini, S. Manzoni (Eds.), AI*IA 2005: Advances in Artificial Intelligence. XIV, 614 pages. 2005.

Vol. 3662: C. Baral, G. Greco, N. Leone, G. Terracina (Eds.), Logic Programming and Nonmonotonic Reasoning. XIII, 454 pages. 2005.

Vol. 3661: T. Panayiotopoulos, J. Gratch, R.S. Aylett, D. Ballin, P. Olivier, T. Rist (Eds.), Intelligent Virtual Agents. XIII, 506 pages. 2005.

Vol. 3658: V. Matoušek, P. Mautner, T. Pavelka (Eds.), Text, Speech and Dialogue. XV, 460 pages. 2005.

Vol. 3651: R. Dale, K.-F. Wong, J. Su, O.Y. Kwong (Eds.), Natural Language Processing – IJCNLP 2005. XXI, 1031 pages. 2005.

Vol. 3642: D. Ślęzak, J. Yao, J.F. Peters, W. Ziarko, X. Hu (Eds.), Rough Sets, Fuzzy Sets, Data Mining, and Granular Computing, Part II. XXIII, 738 pages. 2005.

Vol. 3641: D. Ślęzak, G. Wang, M. Szczuka, I. Düntsch, Y. Yao (Eds.), Rough Sets, Fuzzy Sets, Data Mining, and Granular Computing, Part I. XXIV, 742 pages. 2005.

Vol. 3635: J.R. Winkler, M. Niranjan, N.D. Lawrence (Eds.), Deterministic and Statistical Methods in Machine Learning. VIII, 341 pages. 2005.

Vol. 3632: R. Nieuwenhuis (Ed.), Automated Deduction – CADE-20. XIII, 459 pages. 2005.

Vol. 3630: M.S. Capcarrère, A.A. Freitas, P.J. Bentley, C.G. Johnson, J. Timmis (Eds.), Advances in Artificial Life. XIX, 949 pages. 2005.

Vol. 3626: B. Ganter, G. Stumme, R. Wille (Eds.), Formal Concept Analysis. X, 349 pages. 2005.

Vol. 3625: S. Kramer, B. Pfahringer (Eds.), Inductive Logic Programming. XIII, 427 pages. 2005.

Table of Contents

Session I: Foundations

Session II: Authoring

Session III: Representations

Session IV: Proving

Session V: MKManagement Tools

Session VI: Documents

Session VII: MKM Case Studies

Session VIII: Course Materials

Session IX: Migration

A Proof-Theoretic Approach to Hierarchical Math Library Organization

Kamal Aboul-Hosn and Terese Damhøj Andersen

Department of Computer Science, Cornell University, Ithaca, New York, USA
kamal@cs.cornell.edu, katten@kattens.dk

Abstract. The relationship between theorems and lemmas in mathematical reasoning is often vague. No system exists that formalizes the structure of theorems in a mathematical library. Nevertheless, the decisions we make in creating lemmas provide an inherent hierarchical structure to the statements we prove. In this paper, we develop a formal system that organizes theorems based on *scope*. Lemmas are simply theorems with a local scope. We develop a representation of proofs that captures scope and present a set of proof rules to create and reorganize the scopes of theorems and lemmas. The representation and rules allow systems for formalized mathematics to more accurately reflect the natural structure of mathematical knowledge.

1 Introduction

The relationship between theorems and lemmas in mathematical reasoning is often vague. What makes a statement a lemma, but not a theorem? One might say that a theorem is "more important," but what does it mean for one statement to be "more important" than another? When writing a proof for a theorem, we often create lemmas as a way to break down the complex proof, so perhaps we expect the proofs of lemmas to be shorter than the proofs of theorems. We also create lemmas when we have a statement that we do not expect to last in readers' minds, i.e., it is not the primary result of our work. The way we make these decisions while reasoning provides an inherent hierarchical structure to the set of statements we prove. However, no formal system exists that explicitly organizes proofs into this hierarchy.

Theorem provers such as NuPRL, Coq, and Isabelle provide the ability to create lemmas. But their library structures are flat, and no formal distinction exists between lemmas and theorems [1, 2, 3]. The reasons to distinguish lemmas from theorems in these systems is the same as the reasons in papers: to ascribe various levels of importance and to introduce dependency or scoping relationships.

We seek to formalize these notions and provide a proof-theoretic means by which to organize a set of proofs in a hierarchical fashion that reflects this natural structure. Our thesis is that the qualitative difference between theorems and lemmas is in their *scope*. Scope already applies to mathematical notation. Never in a paper would one need to define the representation of a set ($\{\ldots\}$) nor operators such as union and intersection. Set notation is standard, thus has a

M. Kohlhase (Ed.): MKM 2005, LNAI 3863, pp. 1–16, 2006.

global scope that applies to any proof. However, one often defines operators that are only used for a single paper; the author does not intend for the notation to exist in other papers with the same meaning without being defined again. Similarly, a *theorem* is a statement that can be used in any other proof. Its scope is global, just as set notation. A *lemma* is a statement with a local scope limited to a particular set of proofs. We want a system that represents and manipulates scope formally through the structure of the library of proofs.

In this paper, we propose such a system. First, we propose a formal definition of scoping for proof libraries. Next, we describe a representation of proofs that is able to capture this definition of scope based on work by Kozen and Ramanarayanan [4]. We provide a set of formal rules to create and reorganize the scopes of theorems and lemmas.

We believe that the ability to create and manage complex scoping and dependency relationships among proofs will allow systems for formalized mathematics to more accurately reflect the natural structure of mathematical knowledge.

2 A Motivating Example

Consider reasoning about a Boolean algebra $(B, \vee, \wedge, \neg, 0, 1)$. Boolean algebra is an equational theory, thus contains the axioms of equality:

$$\mathsf{ref} : x = x \tag{1}$$
$$\mathsf{sym} : x = y \to y = x \tag{2}$$
$$\mathsf{trans} : x = y \to y = z \to x = z \tag{3}$$
$$\mathsf{cong}_{\wedge} : x = y \to (z \wedge x) = (z \wedge y) \tag{4}$$
$$\mathsf{cong}_{\vee} : x = y \to (z \vee x) = (z \vee y) \tag{5}$$
$$\mathsf{cong}_{\neg} : x = y \to \neg x = \neg y \tag{6}$$

All variables are implicitly universally quantified in these axioms. Suppose we wanted to prove the following elementary fact:

Theorem 1.

$$\forall a, b, c, z. a = b \to a = c \to z \vee (a \wedge b) = z \vee (a \wedge c) \tag{7}$$

Here is how a proof might go. First, we could prove a lemma.

Lemma 1.

$$\forall x, y, z. x = y \to z \vee (x \wedge x) = z \vee (x \wedge y) \tag{8}$$

Using $a = b$ and $a = c$ from the statement of our theorem, we could apply the lemma under the substitutions $[x/a, y/b, z/z]$ and $[x/a, y/c, z/z]$ to deduce

$$z \vee (a \wedge a) = z \vee (a \wedge b) \tag{9}$$
$$z \vee (a \wedge a) = z \vee (a \wedge c) \tag{10}$$

Next, we know from applying symmetry to (9) that

$$z \vee (a \wedge b) = z \vee (a \wedge a) \tag{11}$$

Finally we conclude from transitivity, (9), and (11) that

$$z \vee (a \wedge b) = z \vee (a \wedge c)$$

which is what our theorem states.

We may decide that (8) does not apply to theorems other than (7), and consequently, should only have a scope limited to the proof of (7). Our representation of proofs makes explicit the limited scope of (8).

Another important observation is that in all places we use (8), the variable z from (7) is always used for the variable z in the lemma. We may wish not to universally quantify z for both (7) and (8) individually, but instead universally quantify z once and for all so that it can be used by both proofs:

$$\begin{aligned} &\forall z,\ \forall a, b, c, a = b \rightarrow a = c \rightarrow z \vee (a \wedge b) = z \vee (a \wedge c) \\ &\quad \text{and } \forall x, y, x = y \rightarrow z \vee (x \wedge x) = z \vee (x \wedge y) \end{aligned} \tag{12}$$

Moving the quantifier for z looks like a simple task, applying the first order logic rule

$$(\forall z.\varphi) \wedge (\forall z.\psi) \equiv \forall z.(\varphi \wedge \psi)$$

However, the proof of the lemma itself must also change, as must any proof that is dependent on this lemma.

Although either version of the lemma can be used to prove the theorem, note that their meanings are subtly different because of the placement of the quantification. Placing a separate quantification of z as in (8) makes the lemma read: "Lemma 1: For all x, y, and z,..." In this case, z is a variable in the lemma for which we expect there to be a substitution whenever the lemma is used in a proof. Using one quantification for both the theorem and the lemma as in (12) makes the lemma read: "Let z be an arbitrary, but fixed boolean value. Lemma 1: For all x and y..." In this case, z is a fixed constant for the lemma.

In this simple example, using (8) or (12) does not matter. However, in other cases, the choices made for quantification may reflect a general style in one's proofs. One may like lemmas to be as general as possible, universally quantifying any variables that appear in the lemma and relying on no constants. On the other hand, one may want to make lemmas as specific as possible, applying only in a select few proofs in order to minimize the number of quantifications. We want to capture this subtle difference formally in our representation of proofs in order to allow the user to choose the representation that best fits the intended meaning.

3 Proof Representation

For representing theorems and lemmas like those in Section 2, we use proof terms similar to those defined in a paper by Kozen and Ramanarayanan [4]. Their

paper presents a *publish-cite* system, which uses proof rules with an explicit library to formalize the representation and reuse of theorems. The system of [4] uses universal Horn equational logic, and we do as well, since it is a good vehicle for illustrating the organization and reuse of theorems. There is no inherent limitation in the system that requires the use of this logic; it could be extended to work with more complex deductive systems.

We use the word "theorem" to mean a theorem, lemma, or axiom. We build theorems from terms and equations. Consider a set of *individual variables* $X = \{x, y, \ldots\}$ and a first-order signature $\Sigma = \{f, g, \ldots\}$. An *individual term* $s, t, \ldots$ is either a variable $x \in X$ or an expression $ft_1 \ldots t_n$, where f is an n-ary function symbol in Σ and $t_1 \ldots t_n$ are individual terms. An equation $d, e, \ldots$ is between two individual terms, such as $s = t$.

A *theorem* is a universally quantified Horn formula of the form

$$\forall x_1, \ldots x_m . \varphi_1 \rightarrow \varphi_2 \rightarrow \cdots \rightarrow \varphi_n \rightarrow \psi \tag{13}$$

where the φ_is are equations representing *premises*, ψ is an equation representing the conclusion, and $x_1 \ldots x_m$ are the variables that occur in the equations $\varphi_1, \ldots, \varphi_n, \psi$. A formula may have zero or more premises. These universally quantified formulas allow arbitrary specialization through term substitution. An example of this is the use of (8) with substitutions to get (9) and (10).

Let $\mathcal{P}$ be a set of *proof variables* $p, q, \ldots$. A proof of a theorem is a λ-term abstracted over both the proof variables for each premise of a theorem proven by the proof and the individual terms that appear in the proof. A *proof term* is:

- a variable $p \in \mathcal{P}$
- a constant, referring to the name of a theorem
- an application $\pi\tau$, where π and τ are proof terms
- an application πt, where π is a proof term and t is an individual term
- an abstraction $\lambda p.\tau$, where p is proof variable and τ is a proof term
- an abstraction $\lambda x.\tau$, where x is an individual variable and τ is a proof term

When creating proof terms, we have the typing rules seen in Table 1. These typing rules are what one would expect for a simply-typed λ-calculus. The typing environment Γ maps variables and constants to types. According to the Curry-Howard Isomorphism, the type of a well-typed λ-term corresponds to a theorem in constructive logic and the λ-term itself is the proof of that theorem [5]. For example, a theorem such as (13) viewed as a type would be realized by a proof term representing a function that takes an arbitrary substitution for the variables x_i and proofs of the premises φ_i and returns a proof of the conclusion ψ.

In [4], a library of theorems is represented as a flat list of proof terms. All of the theorems have global scope, i.e., they are able to be cited in any other proof in the library.

The goal of this paper is to provide a scoping discipline so that naming and use of variables can be localized. The proof term itself should tell us in which proofs we can use a lemma. We use a construct similar to the SML `let` expression, which limits the scope of variables in the same way we wish to limit the scope of lemmas.

Table 1. Typing rules for proof terms

$$\frac{}{\Gamma, p : e \vdash p : e} \qquad \frac{}{\Gamma, c : \varphi \vdash c : \varphi}$$

$$\frac{\Gamma \vdash \pi : e \to \varphi \quad \Gamma \vdash \tau : e}{\Gamma \vdash \pi\tau : \varphi} \qquad \frac{\Gamma \vdash \pi : \forall x.\varphi}{\Gamma \vdash \pi t : \varphi[x/t]}$$

$$\frac{\Gamma, p : e \vdash \tau : \varphi}{\Gamma \vdash \lambda p.\tau : e \to \varphi} \qquad \frac{\Gamma \vdash \tau : \varphi}{\Gamma \vdash \lambda x.\tau : \forall x.\varphi}$$

In order to represent theorems in a hierarchical fashion, we add two kinds of proof terms:

- a sequence $\tau_1; \ldots; \tau_n$, where $\tau_1, \ldots, \tau_n$ are proof terms. This allows several proofs to use the same lemmas. Sequences cannot occur inside applications.
- an expression $\mathsf{let}\ L_1 = \tau_1 \ldots L_n = \tau_n\ \mathsf{in}\ \tau\ \mathsf{end}$. This term is meant to express the definition of a set of lemmas for use in a proof term τ. The τ_is are proof terms, each bound to an identifier L_i. With the existence of the sequences, each τ_i may define the proof for more than one lemma. The identifiers L_i are arrays, where the j^{th} element, denoted $L_i[j]$, is the name of the lemma corresponding to the j^{th} proof in τ_i not bound to a name in τ_i, denoted $\tau_i[j]$. The let expression binds names to the proofs and limits their scope to proof terms that appear later in the let expression. In other words, a lemma $L_i[j]$ can appear in any proof $\tau_k, k > i$, or in τ. The name of a lemma has the same type as the proof to which it corresponds. This scoping discipline for lemmas corresponds exactly to the variable scoping used in SML let expressions.

These new rules have corresponding typing rules, in Table 2.

Table 2. Typing rules for proof terms

$$\frac{\Gamma \vdash \tau_1 : \varphi_1 \quad \ldots \quad \Gamma \vdash \tau_n : \varphi_n}{\Gamma \vdash \tau_1; \ldots; \tau_n : \varphi_1 \wedge \ldots \wedge \varphi_n}$$

$$\frac{\begin{array}{l}\Gamma \vdash \tau_1 : \varphi_1 \\ \Gamma, L_1 : \varphi_1 \vdash \tau_2 : \varphi_2 \\ \ldots \\ \Gamma, L_1 : \varphi_1, \ldots, L_{n-1} : \varphi_{n-1} \vdash \tau_n : \varphi_n \\ \Gamma, L_1 : \varphi_1, \ldots, L_n : \varphi_n \vdash \tau : \varphi\end{array}}{\Gamma \vdash \mathsf{let}\ L_1 = \tau_1 \ldots L_n = \tau_n\ \mathsf{in}\ \tau\ \mathsf{end} : \varphi_1 \to \ldots \to \varphi_n \to \varphi}$$

The rule for a sequence of proof terms is relatively straightforward; the type of a sequence is the conjunction of the types of the proof terms in the sequence. The typing rule for the let expression is based on the scoping of the proofs. We must be able to prove that each proof τ_k has type φ_k under the assumption that all variables $L_i, i < k$ have the type φ_i, where τ_i is assigned to L_i. Finally, we

must be able to prove that τ has the type φ under the assumption that every L_i has type φ_i.

As an example, we represent the proofs of (7) and (8) as

$$\begin{array}{l}\mathsf{thm} = \\ \quad \mathsf{let}\ \mathsf{lem} = \lambda x \lambda y \lambda z \lambda P.(\text{Proof of lemma}) \\ \quad \mathsf{in} \\ \quad\quad \lambda a \lambda b \lambda c \lambda z \lambda Q \lambda R.\mathsf{trans}\ (\mathsf{sym}\ (\mathsf{lem}\ Q))\ \ (\mathsf{lem}\ R) \\ \quad \mathsf{end}\end{array}$$

where thm is the name assigned to (7) and lem is the name assigned to (8). For ease of reading, we have omitted the applications of proof terms to individual terms, which represent the substitution for individual variables. P, Q, and R are proofs of type $x = y$, $a = b$, and $a = c$, respectively.

If we choose to universally quantify z only once as in (12), we represent the proof as

$$\begin{array}{l}\mathsf{thm} = \\ \quad \lambda z.\mathsf{let}\ \mathsf{lem} = \lambda x \lambda y \lambda P.(\text{Proof of lemma}) \\ \quad\quad \mathsf{in} \\ \quad\quad\quad \lambda a \lambda b \lambda c \lambda Q \lambda R.\mathsf{trans}\ (\mathsf{sym}\ (\mathsf{lem}\ Q))\ \ (\mathsf{lem}\ R) \\ \quad\quad \mathsf{end}\end{array}$$

As we can see, there is a one-to-one correspondence between the positions of λ-abstractions and where individual variables are universally quantified. We formally develop the proof terms for thm and lem in Section 5.

4 Proof Rules

We provide several rules for creating and manipulating proofs. The rules allow one to build proofs constructively. They manipulate a structure of the form $\mathcal{L}; \mathcal{C}; \mathcal{T}$, where

- $\mathcal{L}$ is the library of theorems, $T_1 = \pi_1, \ldots, T_n = \pi_n$, where T_i is an array of identifiers with the j^{th} element denoted $T_i[j]$, naming the j^{th} proof in π_i, denoted $\pi_i[j]$,
- $\mathcal{C}$ is the list of lemmas currently in scope, $L_1 = \tau_1, \ldots, L_m = \tau_m$, with components defined as they are for $\mathcal{L}$, and
- $\mathcal{T}$ is a list of annotated *proof tasks* of the form $A \vdash \pi : \varphi$, where A is a list of assumptions, π is a proof term, and φ is an unquantified Horn formula.

In these rules, we use the following notational conventions:

- α and β are proof variables or individual variables.
- $\overline{X}$ is a set of elements $\{X_1, \ldots, X_n\}$, where X_i can be an individual variable or a proof variable.
- $T = \pi$ binds a proof term π to an identifier T. The term π may define the proof for more than one theorem. Therefore, the identifier T is an array,