Dagstuhl Seminar on Deduction

Wolfgang Bibel
Technische Hochschule Darmstadt

Koichi Furukawa
Keio University

Mark Stickel
SRI International

Logic is an essential formalism for computer science and artificial intelligence. It is used in such diverse and important activities as

- Problem specification;
- Program transformation, verification, and synthesis;
- Hardware design and verification;
- Logic programming;
- Deductive databases;
- Knowledge representation, reasoning, diagnosis, and planning;
- Natural language understanding;
- Mathematical theorem proving.

The universality of the language of logic, the certainty about the meaning of statements in logic, and the implementability of operations of logic, all contribute to its usefulness in these endeavors.

Implementations of logical operations are realized in the field of automated deduction, which has introduced fundamental techniques such as unification, resolution, and term rewriting, and developed automated deduction systems for propositional, first-order, higher-order, and non-classical logics.

The 1995 Dagstuhl Seminar on Deduction, succeeding the one in 1993, was convened to give international researchers on deduction the opportunity to meet and discuss techniques, applications and research directions for deduction. It featured 38 presentations, a panel discussion, and an excursion. Some of the results were deemed really exciting by the audience; among those are the discovery of proofs for a number of new mathematical theorems (in algebraic geometry, quasi-groups etc.) which were established with substantial participation of automatic systems; further the mechanical proofs of large parts of mathematical books on set theory, automata theory, and so forth. With the comfortable facilities available at Dagstuhl, system demonstrations were given such as a geometry prover, remarkable not only for its proof power but also for its amazing interface featuring geometrical constructions along with the formal theories and theorems. To an extent limited by the restricted number (43) of participants the seminar provided also a forum for presenting results obtained in a German project (DFG Schwerpunktprogramm) on deduction to the international participants.

The success of this meeting was due in no small part to the Dagstuhl Seminar Center and its staff for creating such a friendly and productive environment. The organizers and participants greatly appreciate their effort. The organizers also thank Uwe Egly for his support in many organizational details. Financial support from NSF and CEU (directly and through Compulog) are also greatly appreciated.

Contents

A Reduction Ordering for Higher-Order Terms

Jürgen Avenhaus, Carlos Loría-Sáenz, and Joachim Steinbach
Universität Kaiserslautern

We investigate one of the classical problems of the theory of term rewriting, namely termination. We present an ordering for comparing higher-order terms that can be utilized for testing termination and decreasingness of higher-order conditional term rewriting systems. The ordering relies on a first-order interpretation of higher-order terms and a suitable extension of the recursive path ordering.

Term rewriting systems (TRSs) can be considered as a powerful theoretical model for reasoning about functional and logic programming in an abstract way, independently of a particular programming language. In such an approach to computer programming, logic and functional programs are represented by means of executable specifications essentially consisting of conditional equations. The operational semantics of these specifications is defined by term rewriting and equation solving, respectively. The extension of first-order logic to higher-order logic by means of (universally quantified) conditional equations enormously increases the expressive power of the specifications and permits an efficient operationalization. In this paper we study how to prove termination of higher-order conditional term rewriting systems (HCTRSs).

In order to verify termination of HCTRSs, higher-order terms must be compared in a suitable ordering. We develop a method for performing that task which is based on an extensions of first-order techniques. More specifically, we construct an ordering called HPO (for *higher-order path ordering*) by means of an extension of the recursive path ordering RPO so that non-algebraic terms can also be compared. Termination and decreasingness of HCTRSs are then achieved as usual in the first-order case: the associated rewrite relation is required to be included in this ordering guaranteeing some kinds of monotonicity properties wrt. term structure and substitutions.

Our method is essentially based on a first-order interpretation of higher-order terms in $\beta\eta$-normal form. Therefore, some of the principal properties and proof-techniques existing in the first-order case can also be used for the HPO.

Combination of Constraint Solving Techniques: An Algebraic Point of View

Franz Baader — RWTH Aachen
Klaus U. Schulz — Universität München

In a previous paper we have introduced a method that allows one to combine decision procedures for unifiability in disjoint equational theories. Lately, it has turned out that the prerequisite for this method to apply—namely that unification with so-called linear constant restrictions is decidable in the single theories—is equivalent to requiring decidability of the positive fragment of the first order theory of the equational theories. Thus, the combination method can also be seen as a tool for combining decision procedures for positive theories of free algebras defined by equational theories.

The present paper uses this observation as the starting point of a more abstract, algebraic

approach to formulating and solving the combination problem. Its contributions are twofold. As a new result, we describe an optimization and an extension of our combination method to the case of constraint solvers that also take relational constraints (such as ordering constraints) into account. The second contribution is a new proof method, which depends on abstract notions and results from universal algebra, as opposed to technical manipulations of terms (such as ordered rewriting, abstraction functions, etc.)

Integrating Automated Deduction and Symbolic Computation Techniques

Leo Bachmair
SUNY - Stony Brook

Theorem proving systems, whether interactive or fully automated, are usually not very efficient at dealing with mathematical domains. One approach to equipping theorem provers with built-in knowledge about specific theories is the integration of decision procedures into general-purpose deductive systems. In this talk, I will discuss the integration of certain algorithms for dealing with ring theory, as they are used in many computer algebra systems, in a resolution-type theorem prover for logic with equality. More specifically, I will show how Buchberger's algorithm for constructing a canonical basis for a polynomial ideal can be reformulated in logical terms, so that algebraic manipulations on polynomials correspond to deductive inferences. Furthermore, it is possible to extract from the polynomial algorithm certain strategies that are useful in guiding the proof search of the deductive system. The resulting combined method can be extended to theories specified by arbitrary sets of clauses, enriched by a commutative ring. Similar techniques are applicable in other contexts as well, such as geometry theorem proving.

(This research resulted from a joint collaboration with Harald Ganzinger, MPI, Saarbruecken, Germany.)

Type Inferencing for First Order Logic with Polymorphic Order-sorted Types

Christoph Beierle
FernUniversität Hagen

The problem of complete type inferencing for polymorphic order-sorted logic programs studied here is as follows: Given a type language with polymorphic order-sorted types and a formula over a signature with typed function and predicate symbols, derive a mapping from variables occurring in the formula to types such that the formula is well-typed. We show that previous approaches are incomplete even if one does not employ the full power of the used type systems. We present a complete type inferencing algorithm that allows for parametric polymorphism as in ML and for hierarchically structured monomorphic types. It can easily be extended to first order predicate logic and also to the case where subtype relationships are also allowed between polymorphic types having the same arity. Various details as well as related aspects of this work are presented in [1], [2] and [3].

[1] C. Beierle and G. Meyer. Run-time type computations in the Warren Abstract Machine. *Journal of Logic Programming*, 18(2):123–148, February 1994.

[2] C. Beierle. Concepts, implementation, and applications of a typed logic programming language. In C. Beierle and L. Plümer, editors, *Logic Programming: Formal Methods and Practical Applications*, Studies in Computer Science and Artificial Intelligence, chapter 5, pages 139–167. Elsevier Science B.V./North-Holland, Amsterdam, Holland, 1995.

[3] C. Beierle. Type inferencing for polymorphic order-sorted logic programs. In L. Sterling, editor, *Proceedings of the Twelfth International Conference on Logic Programming - ICLP'95*, Tokyo, Japan. MIT Press 1995 (to appear).

Decomposition of Tautologies into Regular Formulas and Strong Completeness of Connection-Graph Resolution

Wolfgang Bibel
TH Darmstadt

Elmar Eder
Universität Salzburg

This paper addresses and answers a fundamental question about (ground) resolution. Informally, what is gained with respect to the search for a proof by performing a single resolution step? It is first shown that any unsatisfiable formula may be decomposed into regular formulas provable in linear time (by resolution). A relevant resolution step strictly reduces at least one of the formulas in the decomposition while an irrelevant one does not contribute to the proof in any way. The relevance of this novel insight into the nature of resolution and of the unsatisfiability problem for the development of proof strategies and for complexity considerations are briefly discussed.

The decomposition also provides a novel technique for establishing completeness proofs for refinements of resolution. As a first application, connection-graph resolution is shown to be strongly complete on the ground level. The result (as well as the aforementioned answer) can be lifted to the first-order level the way briefly outlined in the paper. This settles a problem which remained open for two decades despite many proof attempts. The result is relevant for theorem proving because without strong completeness a connection graph resolution prover might run into an infinite loop even on the ground level.

Semantic Resolution, Lemmaizing and Contraction

Maria Paola Bonacina
University of Iowa

Subgoal-reduction strategies, such as those based on Model Elimination and implemented in Prolog Technology Theorem Proving, prevent redundant search by using lemmaizing (caching), whereas contraction-based strategies prevent redundant search by using contraction rules, such as subsumption. In this work we show that lemmaizing and contraction can coexist in the framework of semantic resolution. We define two meta-level inference rules for lemmaizing in semantic resolution, one for unit lemmas and one for non-unit lemmas, and we prove their soundness. Rules for lemmaizing are meta-rules because they use global knowledge about the derivation, e.g. ancestry relations, in order to derive lemmas. Then we define a purity deletion rule for first-order clauses that preserves completeness. Thus we can have a semantic resolution strategy with resolution, factoring, lemmaizing and contraction (subsumption, clausal simplification

and purity deletion). Our meta-rules for lemmaizing generalize to semantic resolution the rules for lemmaizing in Model Elimination. Subsumption and purity deletion in a forward-resoning strategy correspond to success and failure caching, respectively, in a subgoal-reduction strategy.

Towards a Decidable Class of n-ary Horn Predicates

Jochen Burghardt
GMD Berlin

Formalisms to finitely describe infinite sets of ground terms have numerous applications in automated reasoning and other fields of computer science. Existing formalisms are regular tree grammars, semi-linear term declarations, regular substitution sets, various approaches based on set constraints, Ω-terms, ... The minimum requirements to such a formalism is that it should be decidable whether a representation denotes the empty set, and that the intersection of two representable sets is again representable. A first comparison of such formalisms based on their translation into Horn clauses is given, where term sets are described as solution sets of predicates.

A new formalism based on term orderings that commute with the least-common-instance operator (lci) is presented. Define a mapping ϕ from the set of all terms mod. renaming into itself to be a *descending homomorphism* iff $\phi(lci(t_1, t_2)) = lci(\phi(t_1), \phi(t_2))$ and $\phi(t) \leq t$ wrt. some well-founded ordering $<$. We consider the class of Horn program consisting of arbitrary facts and of clauses of the form $p_i(t_i) \leftarrow p_{i1}(\phi(t_i))$ for some descending homomorphism ϕ. (Since neither of the t_i nor of the $\phi(t_i)$ are restricted to be variables, n-ary predicates can be coded e.g. by using "," as a binary function symbol.) Algorithms to decide the satisfiability of any such p_i as well as to compute new clauses for the conjunction of any p_i and $p_{i'}$ are given. While their correctness follows from standard fixed point arguments, their termination relies on the properties of ϕ. Neccessary and sufficient condition for the extendibility of an assignment $t_1 \mapsto t'_1, \ldots, t_m \mapsto t'_m$ to a descending homomorphism are given.

It is conjectured but not yet verified that this approach can be generalized to allow arbitrarily many body literals; in this case, regular tree grammars, regular substitution sets, and semi-linear term declarations would be subsumed.

Automated Production of Human-Readable Proofs in Geometry

Shang-Ching Chou, Xiao-Shan Gao, and Jing-Zhong Zhang
The Wichita State University

The algebraic methods based on the coordinate approach, i.e., Wu's method and the Gröbner basis method, have been very succcesuful in proving hundreds of geometry theorems of equality type. The proofs produced by these methods are generally not readable for two reasons: coordinates do not have clear geometry meanings and polynomials in the proofs are generally large.

We introduce a completely new method, the area method, and the computer program based on this method has produced elegant, short (sometimes even shorter than those given by geometry experts), and human-readable proofs of over 400 theorems. The new method is summarized in the book *Machine Proofs in Geometry* (World Scientific, 1994). Our computer program, GE (Geometry Expert), is now available via ftp at

emcity.cs.twsu.edu: pub/geometry/software/ge_sun.tar.Z.

Structuring Theories and Proofs

Bernd Ingo Dahn
Humboldt University Berlin

The ILF system uses automated theorem provers developed in the DFG-Schwerpunkt "Deduktion" to assist the user in the construction of large formal proofs. The ILF user can support the work of the integrated automated systems by specifying lemmata or by pointing to a specific part of the knowledge base to guide the the solution of a specific problem. In order to do this, the theory has to be structured in an appropriate way. ILF gives the possibility to combine the structuring of theories with the introduction of sorts. Definitions of sorted theories can be parametrized and can include other theories. By specifying in a derived theory an axiom with the same name as in the parent theory, axioms can be overloaded. Using these tools, large knowledge bases can be constructed and structured efficiently.

The interactive ILF system provides the Block Calculus to support the user in editing well-structured proofs. These proofs are well suited for automated and interactive restructuring and flexible presentation. Model elimination proofs generated by the integrated provers SETHEO and KoMeT are automatically translated into the Block Calculus.

A natural language LaTeX presentation of a solution of the Steam Roller Problem by KoMeT was given in order to show the effect of the ILF proof restructuring tools. It was indicated, how the structuring of the theory in use can be exploited in order to adopt the natural language output to the knowledge of a specific reader.

Rewrite-Based Deduction: Expansion and Contraction

Nachum Dershowitz
University of Illinois

At the heart of automated deduction lies an inference engine that repeatedly expands the database of formulæ by applying rules of inference (forwards or backwards). Unlike classical proof theory, the database is not only expanded during deduction, but also contracted as a result of (typically ad-hoc) deletion of formulæ deemed "redundant" (e.g. tautology, subsumption, and demodulation tests).

Research in rewriting has contributed in various ways, including the suggestion of refinements of existing inference calculi. On the conceptual level, notions from rewriting have led to a formalization of contraction steps in theorem provers. A (possibly goal-oriented) complexity measure for proofs can be used to characterize redundant expansions that can be glossed over, as well as contractions to eliminate newly introduced redundancies. This approach provides a methodology for completeness proofs in the presence of contraction. Every proof based on the current formula set that requires a non-redundant inference must contain a subproof that can be replaced—after some expansion step—to obtain a strictly smaller proof. Then contraction does not harm completeness if it never forces the complexity of a proof to increase.

Improved versions of completion (with oriented instances of equations, contextual simplification, and critical pair criteria), resolution (with ordering strategies, subsumption and simplification), Horn-clause deduction (with ordering-based strategies and simplification by decreasing instances), and inductive proofs (with rippling) can be phrased within this framework. The choice of complexity measure dictates the precise contractions that are permissible.

Formalizing Inductive Proofs of Network Algorithms

Amy Felty
AT&T Bell Laboratories

Logical specifications of programs implementing distributed network algorithms are often defined precisely enough to enable a (human) verifier to prove the program's correctness, possibly with the aid of a mechanical proof-checker. However, the topology of the underlying network and its relation to the program, which is crucial for the program's correctness, are not formalized. In this talk, we show how the underlying network can be formally defined by means of induction, and how to reason about network algorithms by structural induction. We demonstrate our techniques on a broadcasting algorithm and an underlying network constituting a binary tree of arbitrary size. We use the theorem prover Coq to fully mechanically check the correctness proof.

An important ongoing goal of this work is to integrate model-checking and theorem proving techniques on a problem domain that could benefit from such integration. Model checkers are highly automated but limited in the size and kind of statements that can be proved. Theorem provers are expressive and powerful but require sophisticated insight and guidance by the user. We discuss initial experiments we have carried out on how a model checker could assist in our broadcasting algorithm correctness proof.

This is joint work with Ramesh Bharadwaj of McMaster University and Frank Stomp of AT&T Bell Laboratories.

More on Quasigroup Open Problems

Masayuki Fujita
Mitsubishi Research Institute, Inc.

1 Introduction

An MGTP (model generation theorem prover) [5] on the Parallel Inference Machine with 256 processors [4] made an obvious breakthrough in deciding some finite quasigroup existence problems by a method of model enumeration, after some tries and a success in China [10] and Australia [7].

Since then, two types of systems, such Mark Stickel's DDPP, an implementation of the Davis-Putnam algorithm, and John Slaney's Finder, a finite-domain constraint based enumeration system, have achieved solutions to harder problems in the domain and contributed to great progress in design theory in discrete mathematics [8]. In this process, the major reasons for new breakthroughs are strongly related to "how to minimize guessing," or how to cleverly choose

Problem	Order	BCCP(time:sec)	Davis Putnam(time:sec)	Look Ahead
QG3	8	236(101)	1037(76)	-
QG5	10	27(75)	38(66)	5
	11	43(225)	136(228)	-
	12	191(950)	443(883)	83

Table 0.1: Failed branches in the search space

the next place to assign a value. Choosing a cell with the least alternative values in the latin square or choosing a literal in one of the least length positive clauses is the common criterion of all systems. And heuristics we found have taken the role of shortening the least length positive clauses. A list of the heuristics and the reasons why they are effective will clarify the advantage of representing the problem as propositions. One reason is that all heuristics are represented declaratively and they are independent from the program and are very easy to verify and modify.

Further investigation was made since then and the following two parts of this talk are an interim report of the continuing research.

1. Experimental results by binary clause closure method [3]
2. Further challenging open problems

2 Experiments

In the research domain of finite constraint solving, look ahead is a well known method to propagate constraints [9]. This method is obviously useful for model finding. This has quite improved the search with the high cost of execution time in the quasigroup application [6]. Because look ahead compute again and again all possible unit consequences of each undecided assignment at every node of the search tree. This is exponentially redundant.

The deductive closure of binary clauses(2-Closure) is not so expensive as look ahead and useful to obtain a similar constraint propagation. Propagation of the constraints by 2-Closure occurs when $a \vee b$ and $a \vee \neg b$ are in the closure.

But the look ahead is not equivalent to this. For example, there is a well known method in the constraint domain that the look ahead simulates. In this method, 'supported' or 'non-supported' assignment corresponds to positive or negative fact respectively. A supported assignment $X_i = a$ by X_j is that all possible assignment of X_j derives $X_i = a$. So all the other possible assignment of Xi are negated by this fact. Non-supported assignment is one of the dual constraint propagation of this situation.

For seeing how the above method work to the quasigroup problems, an experimental program BCCP (binary clause closure program) in klic was created and some results (Table 0.1) were obtained. The results of DDPP and Look Ahead are from [8] and [6].

We can see the look ahead has much smaller size of the search tree than BCCP. However BCCP runs reasonably fast and using a basic data structure instead of stack vectors can make some constant times speedup.

3 More challenging problems

The second part is the introduction of further open problems in this domain. Problems are from the reffered pages of the book [2]. Although the order is comparable to QG1 - QG8 in FSS93, none of them are solved by our method. QG11 was stated 'out of the existing computer's power' in the book.

QG9 (p14)
for order n quasigroups, for max size t of partial transversial $t \leq n-1$. For odd n, $t = n$. No counterexamples are found for any order
(partial transversial: a set of cells of a quasigroup with distinct value and all of them are distinct both in row and column)

QG10 (p15)
Are there three mutually orthogonal LS(10) ?

QG11 (P50)
Are there order 11 row complete lattin square
(complete:all $n(n-1)$ ordered pairs$(A_{i,j}, A_{i,j+1})$ are distinct)

QG12 (P116)
Construct N_∞-LS(16)
(N_∞: for all LS(N) for all $1 < I < N$ no subsquare exists)

QG13 (p117)
are there N_∞ -LS(N) other than $N = 2\ a * 3\ b$?

QG14 (P145)
ISOLS(14,4)(Idempotent Self Orthogonal Latin Square order 14 with a hole of size 4) ?

References

[1] F. Bennett, *Quasigroup Identities and Mendelshon Designs*, **Canadian Journal of Mathematics** 41 (1989), pp. 341–368.

[2] J. Dénes, A.D. Keedwell, ed., *Latin Squares: New Developments in the Theory and Application*, North-Holland, 1991.

[3] A. Gelder and Y. Tsuji, *Satisfiability Testing with More Reasoning and Less Guessing* Preprint, 1994.

[4] H. Fujita and R. Hasegawa, *A Model Generation Theorem Prover in KL1 Using Ramified-Stack Algorithm*, **Proc. of ICLP91,** pp.535-548, 1991.

[5] M. Fujita, J. Slaney & F. Bennett, *Automatic Generation of Some Results in Finite Algebra*, **Proc. International Joint Conference on Artificial Intelligence**, 1993.

[6] Masayuki Fujita, Frank O'Carroll, *Heuristics and more Heuristics:Toward Solving Harder Quasigroup Problems (Extended Abstract)*, **workshop on Automated Reasoning In Algebra, CADE-12, 1994**

[7] J. Slaney, *FINDER, Finite Domain Enumerator: Version 2.0 Notes and Guide*, technical report TR-ARP-1/92, Automated Reasoning Project, Australian National University, 1992.

[8] J., Slaney, M. Fujita, & M. Stickel., *Automated reasoning and exhaustive search: Quasigroup existence problems*,**Computers and Mathematics with Applications**, 1993.

[9] A.K.M., Tan, *Seach Strategies in Finite Constraint Satisfaction*, to appear, 1994 Honours Thesis, Australian National University, 1994.

[10] J. Zhang, *Search for Idempotent Models of Quasigroup Identities*, Typescript, Institute of Software, Academia Sinica, Beijing.

Model Elimination, Logic Programming and Computing Answers

Ulrich Furbach
Universität Koblenz-Landau

We prove that theorem provers using model elimination (ME) can be used as answer complete interpreters for *disjunctive logic programming*. For this, the restart variant of ME with a mechanism for computing answers and the ancestry refinement is introduced. Furthermore, we demonstrate that in the context of *automated theorem proving* it is much more difficult to compute (non-trivial) answers to goals, instead of only proving the *existence* of answers. It holds that resolution with subsumption is *not* answer complete. We consider puzzle examples and give a comparative study of OTTER, SETHEO and our restart model elimination prover PROTEIN.

Locality and Saturation

Harald Ganzinger
Max-Planck-Institut für Informatik

We report on joint work with David Basin about relating locality and saturation. McAllester calls an Horn theory local if whenever a ground clause, called query, is entailed by the theory then it is already entailed by ground instances in which all terms are subterms of some query term. Local entailment problems are decidable in polynomial time. We propose a slightly different concept of locality in which we compare the ground atoms in proofs to the ground atoms in the query with respect to some well-founded ordering. We show that a theory is local in this sense if and only if it is saturated under ordered resolution up to redundancy. With our concept of locality we can describe more complexity classes. We can admit full clauses, not just Horn clauses, and we may attempt to transform non-local presentations into local ones by applying saturation.

Proof Presentation

Xiaorong Huang
Universität des Saarlandes

Jörg Siekmann
DFKI

This talk outlines an implemented system called *PROVERB* that explains machine-found nat-

ural deduction proofs in natural language. Different from earlier works, we pursue a *reconstructive* approach. Based on the observation that natural deduction proofs are at a too low level of abstraction compared with proofs found in mathematical textbooks, we define first the concept of so-called *assertion level* inference rules. Derivations justified by these rules can intuitively be understood as the application of a definition or a theorem. Then an algorithm is introduced that *abstracts* machine-found ND proofs using the assertion level inference rules. Abstracted proofs are then verbalized into natural language by a *presentation module*. The most significant feature of the presentation module is that it combines standard hierarchical text planning and techniques that locally organize argumentative texts based on the derivation relation under the guidance of a focus mechanism.

Parallel Term Rewriting with PaReDuX

Wolfgang Küchlin, Reinhard Bündgen, and Manfred Göbel
Universität Tübingen

We report on the construction of PaReDuX, a parallel term-rewriting and completion system. It is designed for shared memory multi-processors and exploits the fine-grained parallelism provided in a multi-threaded environment. PaReDuX contains modules with a plain Knuth-Bendix completion procedure (ptc), a Peterson-Stickel procedure for AC-theories (pac) and an unfailing completion procedure (puc).

Our goal is to enable the construction of parallel theorem provers in the multi-threaded environment of modern multiprocessor workstations. To this end we are developing a discipline of parallel programming, an efficient portable programming environment, and a prototype implementation to evaluate our concepts.

PaReDuX programs are parallelized versions of the corresponding sequential ReDuX programs. The parallelization takes place in the framework of the PARSAC-2 parallel Computer Algebra system, using the same Virtual S-threads parallelization environment. Virtual S-threads supports the parallel execution of C functions calls which need access to a parallel heap structure for list processing. The environment also provides a highly efficient threads system supporting the simultaneous use of tens of thousands of threads. Together with our programming paradigm of parallel divide-and-conquer it releases the programmer from the burden of scheduling parallel tasks and thinking about the actual hardware underneath.

Our completion procedures are strategy-compliant. They parallelize the inner completion loop while the selection of the best new rule in the outer loop, and hence the completion strategy, is exactly the same as for the sequential algorithm. Both critical pair generation and normalization of critical pairs are done in parallel. For AC-completion, we also use parallelism within the reduction of individual terms. We thus achieve speed-ups of up to a factor of 3.6 on 4 processors.

For more details of this work we refer to our articles in the proceedings of ISSAC'94, PASCO'94, and RTA'95.

Function Introduction in Clause Logic

Alexander Leitsch
TU Vienna
(joint work with Matthias Baaz)

Typically computational calculi are characterized by simple logical syntax and economic principles of inference. Particularly clauses are logic–free forms and all substitutions applied within these calculi are most general unifiers. Thus, in some sense, both syntax and inference are minimalized. This feature is quite beneficial to proof search, but the absence of logical structure can lead to very high proof complexity. On the other hand, full logic calculi like LK and natural deduction, allow the formulation of short and highly structured proofs, but are not suited for proof search. F–extension (function introduction) is a method to produce new clauses by some simple quantifier shifting rule and re–skolemization; this rule is simple and computationally controllable. It is shown that F–extension combined with resolution can simulate the power of the full cut rule, thus resulting in a nonelementary speed–up of proof complexity versus ordinary resolution. Therefore F–extension combines the good features of computational calculi with the strength of full logic calculi. A well–known principle of function introduction is skolemization, although it is usually considered as mere preprocessing. It is demonstrated that the proof complexity of different skolem forms (prenex versus structural) may differ nonelementarily from each other. This shows the importance of creating and preserving logical structure in automated deduction. Particularly structural transformation and controlled cut–introduction (by F–extension) are powerful techniques to increase the power of computational calculi.

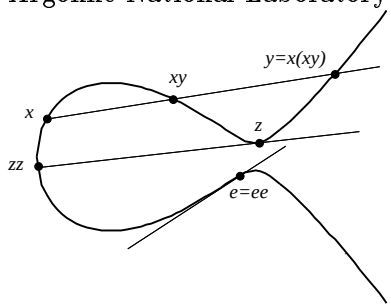
LINUS: A Link Instantiation Prover With Unit Support

Reinhold Letz
TU München

A theorem prover for first-order clause logic is presented which is based on ground instantiation and propositional decision. Like in Plaisted's 1992 approach, the enumeration of (ground) instances is controlled by *hyperlinking*, ie instances of the input clauses are saturated as induced by unification with complementary mates in the increasing formula. The generation of clause instances is supported by an increasing theory of (first-order) unit clauses, which are obtained as a by-product of the hyperlinking process and which are used to simplify the clause instances. The enumeration modulo unit clauses is complete for Horn clause logic, ie, for this sublanguage, no propositional decision procedure is needed. Completeness is preserved under any set-of-support strategy, thus permitting a form of goal-oriented (generalized) unit-resulting resolution. Furthermore, a new method of *dynamic clause splitting* is presented. Dynamic clause splitting minimizes the number of distinct variables in clauses and can reduce the set of generated (ground) instances significantly, without lengthening proofs. The entire prover is implemented in Lisp. Because of the use of indexing techniques, the system is very efficient. The prover was evaluated on the TPTP problem library (v1.1.1), which consists of 2652 formulae (63 % non-Horn, 79 % with equality). The first version of Linus (which has no special equality handling) could solve 39 % of the problems on a Sun Sparc 10, with a time limit of 200 seconds per problem. Thus, the system can compete well with the most successful general purpose theorem provers currently available.

Automated Deduction in Algebraic Geometry

William McCune
Argonne National Laboratory



R. Padmanabhan's inference rule $=(gL)\Rightarrow$ allows one to prove, in an equational setting, some theorems about cubic curves in algebraic geometry without reference to the underlying geometry or topology of the curves. But the rule is awkward for mathematicians to use, and very little had been done with it until we installed it in the automated theorem prover OTTER. Many experiments were conducted, and several new results of interest to researchers in algebraic geometry were obtained by OTTER. Examples of the new results are that cancellative semigroups on a cubic curve must be commutative, and that a 5-ary Steiner law on cubic curve is unique.

Derived Rules for the Generic Reasoning Assistant System EUODHILOS-II

Toshiro Minami
Fujitsu Laboratories Limited

EUODHILOS-II is developed with the intention that it helps us humans with reasoning in various logical structures, especially for new ones. Since it deals with a variety of logics in a uniform way, it is designed to be logic-independent system; the user gives the logic to be dealt with in a NBF-style syntax description and in the Natural Deduction-style logical structures.

Using the representations easy-to-be-recognized by humans and supporting the variety of reasoning styles are fairly important for EUODHILOS-II. Defining and using the derived rules is one of the essentially important features in such an interactive reasoning assistant system like EUODHILOS-II. We proposed a way to improve the present treatment for derived rules in EUODHILOS-II. It intends to overcome the problems of overinstantiation and delay of side condition checking. The idea for the improvement is based on the generalization and dynamic side condition checking. We gave a criterion for classifying the side conditions remains in a derived rule, then gave a way to eliminate redundant conditions.

Implementing Deduction with Constraints

Robert Nieuwenhuis
Technical University Catalonia

We first overview our different theoretical results obtained during the last three years on auto-

mated deduction techniques with constrained equality clauses. After this, their current experimental implementation within the Saturate system is outlined. Its main limitations from the efficiency point of view are analysed, which motivates some ideas on more efficient implementations, based on WAM data structures combined with indexing methods like discrimination- or substitution tree indexing.

Programming Language Semantics in Isabelle

Tobias Nipkow
TU München

In this talk I review a number of different formalizations of language semantics and their equivalence proofs in different logics inside the generic theorem prover Isabelle.

One major case study is the formalization of a simple first-order functional language in LCF. The denotational semantics is given in the standard way, the operational semantics by a step by step simplification, i.e. evaluation at the term level, of the source program. The equivalence proof required 300 lemmas and revealed that LCF complicates matters considerably because it insists that every type is a cpo and that every function is computable and potentially partial.

The second example is the formalization of the first few chapters of Winskel's "The Formal Semantics of Programming Languages" describing the operational, denotational and axiomatic semantics of a simple while-language and their relationship. This was done in 3 different logics: Zermelo-Fraenkel set theory (ZF), Higher-Order Logic (HOL), and HOL enriched with domain theory (HOLCF due to Regensburger). It appears that HOLCF is most suitable for this task, although the example is simple enough all three logics can handle it very easily.

Term Indexing, a Parallel Theorem Prover, More Parallelism by Transformation

Hans Jürgen Ohlbach
Max-Planck-Institut für Informatik

The manipulation of large amounts of formulae in an automated deduction system can be supported considerably by special term indexing mechanisms. A term index is a datastructure representing sets of terms and allowing fast retrieval of particular terms. Typical operations are, given an index I and a term t, find all terms in I which are unifiable with t, instances of t or more general than t. Other operations are: given two indices I and J, compute a third index representing the pairs of unifiable terms in I and J.

We (that means in the first place Peter Graf and Christoph Meyer) have implemented a number of indexing mechanisms, path indexing, discrimination tree indexing, abstraction tree indexing and substitution tree indexing. Substitution tree indexing showed the best performance on a considerable variety of examples. The functionality of the different indexing schemes, however, is different. For example FPA indexing can be implemented such that parallel access of different processes is possible, and even more, one process may change the index while another process does a retrieval.

Most of the indexing schemes we have investigated are implemented in the library ACID,

accessible via FTP from pub/tools/deduction/ACID.

With the indexes as basic datastructures we have implemented a parallel hyperresolution theorem prover for Horn clause logic. Each non-unit Horn clause is represented by a master process and various satellite processes. The master process is responsible for computing from the incoming indexes which represent initial and derived unit clauses a new index containing the compatible unifiers with the corresponding body literals. These unifiers represent new derived unit clauses. The satellite processes do subsumption on the incoming and outgoing substitutions. All operations work directly on the indices, not on the clauses.

The degree of parallelism in this theorem prover depends on the number of non-unit Horn clauses. In order to increase the number of Horn clauses, I presented a transformation on the initial clause set which replaces particular clauses by a certain number of resolvents. For example one can show that the transitivity clause can be eliminated by adding for each positive literal in the other clauses just one resolvent with one of the two premise literals of the transitivity. This method can be generalized to other self resolving clauses. Applied to the condensed detatchmet clause, for example, we obtained improvements of the search behaviour even for sequential theorem provers like Otter.

Theorem Proving with Binary Decision Diagrams

Hiroshi G. Okuno
NTT Basic Research Laboratories

Binary Decision Diagrams (BDDs) are compact representations of boolean functions (propositional logic). We first review BDDs and describe their application to the Magic Square Problem. The problem is encoded as a set of constraints and each constraint is specified by (arithmetic) logic expression. The BDD for all the solutions is constructed by taking logical *and*'s of all the constraints. In constructing BDDs, the order of constraints is critical to avoid combinatorial explosions. Since the resulting BDD contains all the solutions implicitly, the solutions are extracted by enumerating all the paths from the root of BDD to the terminal node 1. However, various capabilities are possible without enumerating the solutions. For example, the theorem that the sum of the inner four squares of the Magic Square of order 4 (4 x 4 chess board) is the same as that of any column, row or diagonal can be proved by constructing a BDD for the theorem.

Since a (standard) BDD contains logical relations between propositional symbols, its size is much larger than that of simple representation of combination sets. Standard BDDs can represent combination sets by logical functions, but such representations are not canonical. We introduce a new kind of BDD called 0-suppressed-BDD (0-sup-BDD) to represent a combination set uniquely. The following two combination-set operations are introduced:

- *Restriction* ($F \triangle C$): $F \triangle C \equiv \{x \in F \mid \exists y \in C \quad x \supseteq y\}$
- *Exclusion* ($F \bigtriangledown C$): $F \bigtriangledown C \equiv F - (F \triangle C)$

Some theorems concerning combination-set operations are used to reorder the order of applying operations to avoid the combinatorial explosions. The main differences of 0-sup-BDD from standard BDDs in solving the Magic Square problems and N-Queen problems are (1) the size of BDD is by ten times smaller, and (2) the same constraints may be applied repeatedly. The

latter property is due to the theorem that elements of constraint C that are irrelevant to F are ignored in restriction and exclusion operations.

Proof Logging and Proof Checking in NEVER

William Pase
ORA Canada

NEVER is the theorem prover for the EVES program specification and verification system. The proof logging and proof checking efforts are intended to increase the level of assurance in the proofs generated by NEVER. The ultimate goal is to place responsibility for the validity of proofs upon a simple, formally verified proof checker.

The proof logging effort will result in a version of NEVER that can produce detailed descriptions of the generated proofs in the form of a list of the inferences. These proof logs will be used for proof checking and proof browsing. Each of the inferences must be simple and easy to check, including those generated by decision procedures.

The proof checking effort will result in a formally verified proof checker. This will provide a high degree of assurance in the proofs generated, independent of the complexity of the NEVER theorem prover. Because of the simplicity of the inferences in the proof logs, checking does not involve search, it consists of stepping through the inferences.

The proof browsing effort will result in a proof browser that will allow a proof to be displayed while controlling the amount of detail. Browsing can be used for the validation of proofs, and for the understanding of proofs. This requires that proof logs contain structure, in the form of annotations, in addition to the inferences.

There is an experimental version of NEVER that generates proof logs for everything except simplification by decision procedures. However, this is being implemented. There is a prototype proof checker that has successfully checked all of the proofs within the EVES test suite. The proof browser is being designed.

Mechanising Set Theory: Cardinal Arithmetic and the Axiom of Choice

Lawrence C. Paulson
University of Cambridge

Krzysztof Grąbczewski
Nicholas Copernicus University

Fairly deep results of Zermelo-Fraenkel (ZF) set theory have been mechanised using the proof assistant Isabelle. The results concern cardinal arithmetic and the Axiom of Choice (AC). A key result about cardinal multiplication is $\kappa \otimes \kappa = \kappa$, where κ is any infinite cardinal. Proving this result required developing theories of orders, order-isomorphisms, order types, ordinal arithmetic, cardinals, etc.; this covers most of Kunen, *Set Theory*, Chapter I. Furthermore, we have proved the equivalence of 7 formulations of the Well-ordering Theorem and 20 formulations of AC; this covers the first two chapters of Rubin and Rubin, *Equivalents of the Axiom of Choice*. The definitions used in the proofs are faithful in style to the original mathematics.

Reasoning Theories: Towards an Architecture for Open Mechanized Reasoning Systems

Fausto Giunchiglia
IRST and Università di Trento

Paolo Pecchiari
IRST and Università di Genova

Carolyn Talcott
Stanford University

Our ultimate goal is to provide a framework and a methodology which will allow users, and not only system developers, to construct complex systems by composing existing modules, or to add new modules to existing systems, in a "plug and play" manner. These modules and systems might be based on different logics; have different domain models; use different vocabularies and data structures; use different reasoning strategies; and have different interaction capabilities. The work presented in this talk, which is a first small step towards our goal, makes two main contributions. First, it proposes a general architecture for a class of reasoning modules and systems called *Open Mechanized Reasoning Systems* (*OMRSs*). An OMRS has three components: a *reasoning theory* component which is the counterpart of the logical notion of formal system, a *control* component which consists of a set of inference strategies, and an *interaction* component which provides an OMRS with the capability of interacting with other systems, including OMRSs and human users. Second, it develops the theory underlying the reasoning theory component. This development is motivated by an analysis of the Boyer-Moore system, NQTHM.

A Flexible Theorem Prover

Uwe Petermann
HTWK Leipzig

The aim of this research (a joint work with Gerd Neugebauer) is to combine the high inference rates of a PTTP-based theorem prover with high flexibility. By flexibility we mean something beyond the manipulation of dozens of switches. In particular we would like to have the possibility to modify the underlying calculus. This includes the possibility to specify the inference rules that will be applied by the prover. Moreover, experience showed that one needs access to data structures maintained by the prover during the proof search. Clearly, the access to those data structures should be save and well understood in terms of the considered calculus.

This idea has been realized by the calculi programming interface CaPrI of the theorem prover ProCom. Inference rules may be specified by a description language. Basic data structures, like the current path maintained by a model elimination prover, are specified as an abstract data type. The user may substitute the default implementation by his favored implementation.

Those features of CaPrI are illustrated by an analysis of three different calculi: a simple kind of theory reasoning, the translation of multi-modal logic into first-order logic and the implementation of lemma application.

The prover is available by ftp. For more informations see [1] and under

`ftp://upsilon.imn.th-leipzig.de/pub/WWW/welcome-german.html`

References

[1] G. Neugebauer and U. Petermann, *Specifications of Inference Rules and Their Automatic*

Translation, Proceedings of the Workshop on Theorem Proving with Analytic Tableaux and Related Methods, 1995.

On Words with Variables

Michakl Rusinowitch
INRIA-Lorraine and CRIN

Ground reducibility is a key property used by inductive completion procedures in order to detect false conjectures. A term is said to be ground reducible whan all its ground instances are reducible. We present the case of word rewrite systems where ground reducibility is undecidable in general and co-NP-complete for the case of a linear system with a linear term. The problem is related to some old open questions in language theory concerning the unavoidability of patterns. The simpler reducibility problem of a word by a (linear) word rewrite system is also related to string matching with variable-length don't cares symbols. This is joint work with **Gregory Kucherov**.

Program Extraction from Classical Proofs

Helmut Schwichtenberg
Universität München

As is well known a proof of a $\forall\exists$–theorem with a quantifier–free kernel — where $\exists$ is viewed as defined by $\neg\forall\neg$ — can be used as a program. We describe a "direct method" to use such a proof as a program, and compare it with Harvey Friedman's A–translation followed by the well–known program extraction from constructive proofs.

A refinement of Harvey Friedman's A–translation is introduced, in order to simplify the extracted program. The simplification concerns the type of the auxiliary functions as well as their if–then–else structure. The type reduction is achieved by not replacing all atoms P by $(P \to A) \to A$. To reduce case splitting we construct "good" proofs of $C \to C^A$ for quantifier–free C. The following example is used to show that such improvements are indeed necessary. Let $f\colon N \to N$ be an unbounded function with $f(0) = 0$. Then one can extract a program from a classical proof of $\forall n \exists m. f(m) \leq n < f(m+1)$. If for instance $f(m) = m^2$, then this formula expresses the existence of an integer square root.

PVS = Decision Procedures + Interaction

Natarajan Shankar
SRI International

SRI's PVS verification system is:

1. An experiment aimed at studying the synergistic interaction between expressive specification language features and powerful deductive capabilities.

2. An attempt to build an interactive proof checker on top of efficient decision procedures for equality, arithmetic, and propositional simplification.

The PVS specification language includes parametric theories, predicate subtypes, dependent types, subtyping judgements, and user-definable abstract datatypes. With these features, typechecking becomes undecidable — the typechecker generates proof obligations that can be discharged using the proof checker. The PVS proof checker contains primitive commands for inference steps like simplification, rewriting, and beta-reduction. Higher-level inference commands can be defined using a simple strategy language with constructs for branching, backtracking, and recursion. PVS has been used in the verification of a commercial microprocessor design where it revealed both seeded and unseeded errors, and in a variety of other large and small verification efforts. PVS only makes limited use of currently available theorem proving technology, and it would be interesting to see how other general and special-purpose theorem proving tools can be integrated into the system.

Two Approaches for Finite-Domain Constraint Satisfaction Problems

Yasuyuki Shirai and Ryuzo Hasegawa
Institute For New Generation Computer Technology

We have developed two types of systems; CP and CMGTP, for finite-domain constraint satisfaction problems. CP is based on the constraint logic programming scheme, and is written in SICStus Prolog. CP has achieved high performance on quasigroup (QG) existence problems in terms of the number of branches and execution time. CP succeeded in solving a new open quasigroup problem. On the other hand, CMGTP is a slightly modified version of our theorem prover MGTP (Model Generation Theorem Prover), enabling negative constraint propagation using the unit simplification mechanism. CMGTP has exhibited the same pruning ability as CP for QG problems. CMGTP can be used as a general constraint solver for finite-domains on which we can write down constraint propagation rules with CMGTP input clauses directly. We also show the methods of parallelization in CMGTP system and the results on the parallel inference machine PIM/m which has 256 processors. We obtained almost linear speedups for QG5.13.

Constraint Satisfaction and Deduction: Some Techniques

John Slaney
Australian National University

Finite domain constraint satisfaction problems (CSPs) are easily described in terms of propositional satisfiability and it is well known that backtracking search methods for CSPs amount essentially to proof searches in propositional logic. In this talk we note several techniques for improving efficiency, drawing on experience with CSP solvers. The two most important are

- the addition of secondary (derived) constraints during the search. These record information extracted from the points at which backtracking was forced, thus making the constraint network contain explicitly what was implicit knowledge about the problem
- lazy constraint generation, whereby first order clauses are not reduced to their ground instances over the domain but instead are kept as first order conditions against which model candidates are tested. If a model candidate fails the test, one violated ground instance of the clause is selected and added to the constraint network. In this way, only a

few ground clauses are used, but these suffice to determine the solutions.

Some sample problems are examined briefly and their characteristics with respect to the suggested techniques are noted.

Equational Reasoning about Quasigroups

Mark E. Stickel
SRI International

Finite quasigroups in the form of Latin squares have been extensively studied in design theory. Some quasigroups satisfy constraints in the form of equations, called *quasigroup identities*. Numerous open problems of the existence of quasigroups of particular size that satisfy particular identities have been solved by automated theorem-proving methods (such as the Davis-Putnam procedure) that are complete over a finite domain. We illustrate how other kinds of questions concerning quasigroup identities can sometimes be answered by the alternative equality-based automated theorem-proving method of term rewriting and completion.

Typelab: Towards an Interactive Prover for Type Theory

Friedrich W. von Henke
Universität Ulm

In this talk we give an overview of the capabilities of the system Typelab currently under development at the University of Ulm. The formal basis of Typelab is a constructive type theory, the Extended Calculus of Construction augmented by inductive types. Into the type theory, the intuitionistic predicate calculus can be embedded. For this calculus, an interactive prover has been developed; it is built on a Gentzen-style sequent calculus, using prover commands and tactics similar to those provided by the PVS prover for classical predicate calculus. The prover also incorporates a decision procedure for propositional logic and a proof search procedure for (intuitionistic) predicate logic that generates some proofs automatically. This prover component provides the basic proof capability that supports the other more experimental ones. A second approach investigated in Typelab involves representing theories in a hierarchy similar to concept hierarchies in terminological logics, the idea being that a theorem would be stated and proved in the "most general" theory in which they it is true; it could then be "inherited" by theories subsumed by that theory. This approach makes use of the fact that theories may be modeled in the type theory as certain kinds of dependent types and theory morphisms, representing a general kind of subsumption relation, can be expressed as mappings between such types. A third kind of reasoning investigated in Typelab is the modeling of meta-reasoning. For this purpose, suitable reflection principles have been designed and implemented; their practical usefulness is currently being explored.

Specification and Analysis of Proof-Valued Computations

Lincoln A. Wallen
University of Oxford

Proof-procedures compute evidence for logical consequences but do not do so simply by enumerating proofs. Techniques such as unification, resolution and its refinements, connection calculi and other analytic methods, all make use of logical structure to improve computational behaviour. The relationships between these methods and techniques are difficult to formulate and this in turn makes it difficult to adapt them to new logical systems and problem domains.

In this talk we outline an approach to the specification and analysis of proof-procedures. We use a metalanguage — the Edinburgh Logical Framework (LF) — to characterise the terms, formulae and proofs of a logical system as particular sets of typed lambda terms via a signature defining the logic. Proof-procedures and their component techniques are then characterised by stating the invariants that they maintain by means of a theory of *partial objects:* approximations to proofs and terms where some structure remains indeterminate. For example: (1) the use of unification and skolemisation within a first-order logic is given an interpretation via partial proofs with indeterminate terms and indeterminate placements of the rules governing quantifiers; (2) the use of connections/clashes and propositional normal forms is given an interpretation via partial proofs whose propositional structure is highly indeterminate. This abstract view enables these techniques to be applied to classes of logical systems defined via their LF signatures, thus generalising standard and recent results extending proof procedures to non-classical logics.

Apart from analysis and theoretical extension, the interpretation supports the compilation of (invariants for) algorithms such as unification from (invariants for) a unification algorithm for the framework language by making use of properties of the signature defining terms. The appropriate invariants for both first-order, and higher-order unification algorithms have been generated in this way from a new presentation of unification for the LF.

In the long run, we hope that this approach will enable us to give a mathematical treatment of *search spaces,* and describe the effect particular refinements have on such spaces.

This is joint work with David Pym (Birmingham), Eike Ritter and Jason Brown (Oxford).

Reusing Proofs

Christoph Walther
TH Darmstadt
(joint work with Thomas Kolbe)

We investigate the application of machine learning paradigms in automated reasoning, viz. the improvement of theorem provers by reusing previously computed proofs. If the prover has computed a proof of a conjecture, the proof is analyzed yielding a so-called *catch.* The catch provides the features of the proof which are relevant for reusing it in subsequent verification tasks and may also suggest useful lemmata. Proof analysis techniques for computing the catch are presented. A catch is generalized in a certain sense for increasing the reusability of proofs. We discuss problems arising when learning from proofs and illustrate our method by several examples.

Model Generation by Propositional Reasoning

Hantao Zhang
The University of Iowa

In the recent two years, we have solved several hundreds of open cases in finite mathematics by a propositional theorem prover for model generation. The contribution of our work is twofold: (1) We provided solutions that mathematicians are interested but cannot obtain either by hand or by special purpose programs; (2) We improved substantially our reasoning tools so that they can be used to attack problems in other domains. Our propositional theorem prover is called SATO (SAtisfiability Testing Optimized) that is based on the Davis-Putnam algorithm. the Davis-Putnam algorithm. The new features of our implementation of the Davis-Putnam algorithm include (i) a novel unit-propagation algorithm and (ii) the trie data structure for propositional clauses.

Abstract of Panel Discussion at Dagstuhl Meeting on Deduction: Mining Proof Attempts for Logical Structure

Lincoln A. Wallen
University of Oxford

A number of speakers in the early part of the Dagstuhl meeting described techniques for adjusting the behaviour of theorem provers over a series of proof attempts. The techniques ranged from the very formal—eg., C. Walther's use of proof-schemata abstracted from successful proofs to guide the search for proofs of new formulae—to the informal—eg., W. McCune's description of parameter setting over successive runs of Otter. A discussion was held to explore the extent to which these techniques could be seen as a way of introducing appropriate *cut-formulae* into a problem so as to bring a proof into the search space at a particular depth, a method used to good effect in A. Leitsch's technique of function introduction to shorten proofs.

The well-known proof-theoretic properties of cut-introduction for shortening proofs was reviewed and the discussion identified following two central questions:

- first, the availability of logical languages suitable to describe the guidance information in particular cases;
- secondly, the problem of describing and measuring the effect of refinements on search spaces.

To use Walther's example as an illustration, the proof schema "mined" from a successful search is expressed in what resembles a second-order language. Formulation of such techniques using logical methods, such as higher-order languages, might permit a logical analysis of the worth of the techniques in shortening the length of shortest proof available.

However, it is a reduction in the *size* of space searched that indicates a practical improvement in theorem proving method. Once again the woeful lack of a theory of search spaces was keenly felt by the participants. As a consequence, effective techniques for improving performance in practice, did not seem to be amenable to analysis simply on the basis of "shortest proof".

In summary, two ideas emerged from the discussion. The need for analytic techniques to

describe and measure the behaviour of theorem proving systems over several proof-attempts (a form of iterative analysis similar to that performed in the machine-learning domain?), and the related need to develop theories of search spaces. The discussion revealed the central place that techniques for the analysis of failed proof-attempts occupies in successful provers (Otter, Boyer-Moore etc.) and could be seen as sounding a call for more analysis and formal description of these sometimes interactive, sometimes automatic, methods for the productive use of failure. The calculation of appropriate cut formulae was identified as an important goal for the future, but, without an analysis of search spaces, would not in itself be sufficient to describe the effect of these "data mining" techniques.