MATHEMATISCHES FORSCHUNGSINSTITUT OBERWOLFACH

T a g u n g s b e r i c h t 3|1976

Automatisches Beweisen

5. 1. bis 10. 1. 1976

Die Tagung über "Automatisches Beweisen" vom 5. 1. bis 10. 1. 1976 unter der Leitung von Prof. Bledsoe (Austin) und Prof. Richter (Aachen) war das erste Treffen dieser Fachrichtung im Mathematischen Forschungsinstitut Oberwolfach und brachte führende Vertreter dieses Gebiets zusammen, darunter 12 Teilnehmer aus Nordamerika. In den Referaten wurden Logik und Beweistheorie behandelt, Resolutions- und Paramodulationstechniken sowie schnelle Algorithmen und Heuristiken vorgeführt, über existierende Programme berichtet und Anwendungen, z.B. in der Programmverifikation, angegeben. An den letzten beiden Abenden fanden abschließende Diskussionen über die wichtigsten offenen Probleme und richtungsweisende Ansätze statt.

Teilnehmer

P.Altmann, Aachen
W.Bibel, München
W.W.Bledsoe, Austin
E.Börger, Münster
F.M.Brown, Edinburgh
A.Bundy, Edinburgh
J.L.Darlington, Bonn
M.Davis, New York
T.Gergely, Budapest
U.Grude, Berlin
P.J.Hayes, Essex
L.Henschen, Evanston
G.Hoffmann, Tübingen
G.Huet, Rocquencourt
W.H.Joyner, Yorktown
K.Justen, Aachen
R.Kowalski, London
D.S.Lankford, Georgetown
A.J.Nevins, Atlanta
R.Overbeek, DeKalb
T.Pietrzykowski, Waterloo
R.Reiter, Vancouver
M.M.Richter, Aachen
J.A.Robinson, Syracuse
C.Schippang, Aachen
W.Schönfeld, Stuttgart
H.K.Seeland, Stuttgart
S.Sickel, Santa Cruz
J.Siekmann, Karlsruhe
S.Tärnlund, Stockholm
M.Tyson, Austin
G.Veenker, Bonn
G.Winterstein, Kaiserslautern

Vortragsauszüge

W.BIBEL: A Syntactic Connection between Proof Procedures and Refutation Procedures

Theoretically, theorem proving can be done by proof procedures as well as by refutation procedures. Practically, almost all procedures are of the latter type because people feel that most problems are of a form close to disjunctive normal form. Now it is shown that the improvement of both types of procedures by avoiding transformation to any normal form results in procedures which under the same search strategy and the same first order technique (either Skolem-function or relational technique) are distinguished only by a trivial syntactic difference. This together with the result already known before that both techniques are equivalent in either type of procedure implies a very close syntactic connection between both types of procedures in the sense that any procedure of one type can easily transformed constructively into one of the other type. On a whole this says that both types are equally appropiate for theorem proving; but it is argued that proof procedures are slightly more natural and can easily be understand and explore.

W.W.BLEDSOE: Introduction to the Conference

I will briefly discuss the history of Automatic Theorem Proving (ATP) from Herbrand's original work until the present. I will then offer some questions which I hope will be answered during the conference or at the review session on Friday night:
Where are we ? (state of the art)
What are the relationships among the different activities in ATP?
What are some of the outstanding problems?
Where are we going?
What should we be doing?

F.M.BROWN: Research Program for the Automation of Mathematical Reasoning

Research towards the automation of mathematical reasoning has stagnated. Although great improvements have been made in the construction of theorem provers for particular mathematical domains such as pure LISP, logic, arithmetic, set theory, topology, analysis and elementary algebra, research along these lines does not in itself explain the mathematical ability to create improved deductive systems. In particular, such research provides no explanation as to how domain dependent mathematical knowledge is acquired or even justified.

We believe that the ability of a mathematical system to improve its deductive capabilities by the acquisition of more sophisticated mathematical techniques is a prerequisite of realistic mathematical reasoning. In this paper we outline a research programme aimed at achieving this goal.

A.BUNDY: Solving Mechanics Problems

We give an overview of current work at Edinburgh on solving mechanics problems. In particular we discuss "The Basic Method" for solving general, non-differentiable equations and Welham's PROLOG program which is based on it. One conclusion of this work is that rewrite rules are a powerful proof technique, but we need systems of such rules, each one of which has an associated test for deciding when it is useful.

J.L.DARLINGTON: Special applications of higher-order theorem proving

A resolution-based theorem prover incorporating a limited higher-order unification algorithm has been applied to the automatic synthesis of programs and to the proof of theorems in general topology. In the automatic programming, the higher-order facility handles the functional and predicate variables that occur in the verification conditions for loops and in axioms for composition and equality substitution, and the prover synthesises partially correct programs that may contain nested loops. In topology, the program uses higher-order unification to instantiate set variables in proving theorems that first-order provers have proved only after instantiation of these variables by hand. Further applications of higher-order theorem proving, such as program verification, are also indicated.

M.DAVIS: A PL/I Implementation of a Language for Theorem Proving

The development of mathematical logic provides a rational reconstruction of the social activity which is mathematical research. The task for theorem-proving is to obtain feasible algorithms by appropiate analysis. Our understanding of how outstanding mathematicans function is far too limited to permit sucessful imitation by computer systems. Of course the problem is difficult, and a genuine solution is not to be expected soon. The discovery of the technique of seeking unification of complementary literals is important: For the first time a general technique permitted automatic selection of the key lemma

in a mathematical argument. Completeness is of secondary importance since no implemented procedure can really be complete.
I'm especially interested in procedures, like linked conjunct strategies which conserve space and can be run for a long time. The language being implemented is a collection of PL/I procedures operating on clauses (represented as arrays of character-strings) with unification as a key primitive. Preliminary tests are encouraging.

T.GERGELY: Theorem Proving by Analogy in "Natural" Mathematical Theories

To develop theorem provers for some mathematical fields we need
1) a formal language of representation of the theorems to be proved and of the necessary information from the corresponding mathematical field. This language must be close to the natural mathematical language
2) analogies which describe the connection between some different fields and theorem proving methods constructed on their bases.
So as to elaborate this theorem prover the general proof methods developed for elementary theories within the frames of first order logic cannot be directly or indirectly used. It is for them necessary to develop a general approach which can give a base of theorem prover oriented to a fixed field. Our study is sentenced to such an approach based on the category theory. It gives a general frame to develop
1. a suitable formal language of mathematical text representation close to the intuition of mathematicians;
2. the exact definitions of different concepts of analogy and of their role in proof discovery. In this study special attention will be paid to the investigation of model theoretic aspects.

U.GRUDE: Design of a Programming Language for Resolution-oriented Theorem Provers

A highly problem-oriented language for easy and structured programming of theorem-proving algorithm is presented. The class of programs that can be written in this language is characterized informally by a (fairly general) model for TP-programs. The model and the language provide special means for the natural and easy formulation of search-strategies. An extendible number of inference rules and strategies are built in or supported by standard definitions.

Important object-types introduced are: infinite sequence, lists and n-tuples. Operations on such objects are: union, product, formation and atomizing. These concepts permit the description of infinite structured objects, the static aspect of a TP-program is stressed and instead of the usual control-structures value-directed, quasi-parallel computation is employed. The language is openly embedded into SIMULA.

P.HAYES: Cleaning up Resolution Theory

Many results in the metatheory of resolution-type inference systems can be established in a more general setting.
1) Assuming only four simple properties (finiteness, recursiveness, conservativeness, non-creativity) of inference rules, the Rabin-Ehrenfeucht lemma can be established.
2) Assuming in addition that the objects in the search space have certain selector functions defined on them, which identify occurrences of atoms, a class of rules can be defined (by partitions) and the lifting lemma established, which applies to all such rules.
3) A particular type of objects, partially closed semantic trees, can be used to define simple search spaces which have as homomorphisms many known inference systems, including resolution, hyperresolution, SL-resolution, model elimination and matrix reduction; as well as some new systems which generalize some of these. The metatheory of all these systems is thus considerably simplified and unified, and many formerly awkward proofs become trivial.

L.HENSCHEN: Experiments with a Theorem-proving Language

I will describe a theorem-proving language implemented at Argonne National Lab. The main features are that the primitives of the language are approximately the same operations and properties of clauses as researchers use in describing theorem-proving strategies and that the language is designed to be extensible. The language and system allow considerably more control by the user of the search than previous systems. The system is implemented in assembly language and is therefore many times faster and has much more storage available than most LISP based systems; it is hoped that this speed and storage capacity will enable harder problems to be attacked and solved than have previously been attempted.
I will also describe some of the strategies that we have been able to program and compare their performance on several problems.

G.HOFFMANN: On the Implementation of Proof Procedures

The implementation of two different proof procedures on the same computer are compared with respect to data structures, programming languages, efficiency, time for implementation and time needed for changes in the programs. The first proof procedure is the unit-clause procedure with built-in equality (cf. Computing, 1971), the second a resolution type procedure for ω-logic (cf. Pietrzykowski, Huet et al.). The conclusion drawn from the comparison is that there should be an efficient, portable, high-level language of LISP-type especially built for theorem-proving (TP) and that there should be a medium for publishing programming methods and algorithms especially suited for TP.

G.HUET: Simplification Sets

Simplification sets will be presented as sets of rewriting rules over a term language, possessing various properties (Church-Rosser property, well foundedness, etc.). Various problems are discussed:
- recognition of these properties
- their mutual relationships
- how to complete a given set in order to possess these properties
- how to apply these simplifications, in particular matching problems
- relevance of these results for automatic theorem proving, formal semantics of programming languages and program optimizations.

W.H.JOYNER: A Resolution Analog of the Herbrand Theorem

The soundness and completeness of resolution strategies are most often proved by establishing the connection between a derivation of the empty clause from a set of clauses and the unsatisfiablity of that set. Here, resolution soundness and completeness are demonstrated in a constructive way, by relating the resolution refutation to a proof in a formal system rather than to the nonconstructive notion of validity. It is shown that from a resolution refutation of the negation of a formula, a proof of the formula in a Hilbert-type system can be constructed primitive recursively, and conversely that the refutation can be constructed primitive recursively from the proof. This is an analog, in resolution terms, of the fundamental theorem of Herbrand's thesis, and a stronger result than the Skolem-Herbrand-Gödel theorem more familiar in automatic theorem proving research.

K.JUSTEN/C.SCHIPPANG: On the Inverse Method

Other than resolution, which tests formulas for unsatisfiability, the inverse method - due to Maslow - tests formulas of the first-order predicate calculus for deducibility. It is essentially a coding of some normed deduction search trees (NDST's) for formulas F in prenex conjunctive normal form in some Gentzen calculus M. NDST's are constructed by starting from F and applying some complex rule R, which is a combination of the rules in M from bottom to top. Branches of a NDST are coded by collections. The basic notion of the inverse method is that of a favorable collection, which is given inductively by the rules A and B. The collections favorable according to A correspond to the branches of a normed deduction tree of F. Rule B corresponds to an application of rule R. So the empty collection is favorable iff F is deducible in M. The inverse p-method, based on shortened collections, is described and the equivalence between it and the inverse method is shown.

R.KOWALSKI: Logic for Problem Solving

Different kinds of resolution rules can be regarded as problem-solving strategies. Hyperresolution supported by the hypotheses of the conclusion is problem-dependent forward chaining. Linear resolution with various redundancy-suppressing ordering strategies is backward chaining from the goal. Various ways of combining bottom-up (forward chaining) and top-down (backward chaining) can be indicated by turning the links of a connection graph into directed arcs which control the allocation of problem solving activity. Problems from the areas of programming, data bases, and human problem-solving are formulated in alternative ways. Different theorem-proving strategies applied to these representations are evaluated for their reasonableness as problem-solvers.

D.S.LANKFORD: Canonical Inference

We establish some new refutation completeness results for sets of rewrite rules in conjunction with resolution and paramodulation. All results of this paper deal with the case when none of the equations of an equality unsatisfiable set occur in non-unit clauses. When the set of reductions is complete we show that blocked resolution and immediate narrowing are refutation complete. We also show that

special paramodulation, which is paramodulation into positions which are not variables, and resolution are refutation complete. Finally we show that, in the presence of a suitable complexity measure, derived reduction is refutation complete. In addition, we draw a connection between complexity measure and decision procedures for elementary algebra.

A.J.NEVINS: Automatic Theorem Proving and Artificial Intelligence

An argument is presented which advocates that research in automatic theorem proving should place greater emphasis on the development of pattern recognition systems which understand their domains rather than upon systems centered primarily around heuristic search. Such research would concern itself more with questions like how should a computer adapt a method which worked on one problem so that it can be used on another problem and what framework would best facilitate the absorption of new knowledge into a problem-solving system.

R.OVERBEEK: An Approach to the Implementation of Theorem-Proving Systems

(In cooperation with E.McCharen and J.McCharen)

This talk presents an approach to the implementation of automated theorem-proving algorithms which has resulted in one of the more sucessful of the existing programs. This approach was formulated and implemented by a team which included the authors at Northern Illinois University. Because the basic architecture of that system differs radically from all others that the authors have examined, and because the initial results which have been obtained by the program have been quite encouraging, we feel that such an exposition will be of interest to other groups actually involved in the implementation of automated theorem-proving programs.

T.PIETRZYKOWSKI: Mechanical Generalization and Semantic Distance

Let K,G and S be conjunctions of skolemized clauses. We shall say that G is a generalization of S over K (knowledge base) iff $(K \wedge G) \supset S$, $K \wedge G$ is satisfiable and G satisfies also some maximality and minimality constraints which prevent possible pathologies and

trivialities. Noticing that the above definition establishes a partial ordering ($G \geq_K S$) we may define a degree of generality $\delta(G,S)$ as the length of the shortest path in the graph generated by $\geq_K$ relation. Consequently we can define the distance $D(S_1,S_2)$ between arbitrary S_1,S_2 as $\min_G(\phi(\delta(G,S_1)) + \phi(\delta(G,S_2)))$ where ϕ is some strictly monotonicaly growing function. These definitions of generality and semantic distance seem to capture some of the intuitions related to these concepts.

R.REITER: On Knowledge Based Formal Reasoning

My talk focuses on a variety of ways that domain specific knowledge can be exploited in so-called natural or "human-oriented" theorem-proving systems. Typically, such systems rely on forward and backward chaining, which are major sources of combinatorial explosion of the search space. Often, domain specific knowledge may be used to dramatically reduce this explosion. Backchaining can be controled through the use of models, and by associating advice with axioms. Forward chaining can be reduced by algebraic simplification routines, by exploiting symmetries in the axioms, and by "compiling in" knowledge about the results of forward chaining into certain axioms. These techniques will be illustrated with examples from my MATH-HACK system, a LISP implementation of a natural deduction theorem prover.

M.M.RICHTER: Equality Logic and Theorem Proving

The (constructive) transformation between proofs in Gentzen-type systems on the one hand and the resolution-paramodulation-type systems on the other side is discussed. This translation is difficult when equality is present. For the propositional part one proceeds by induction on the complexity of the formula; the step to the general level is done essentially by the lifting lemma. Because the lifting lemma fails for ordinary paramodulation one has first to transform a paramodulation derivation into a very special one for which the lifting lemma does hold. This is closely related to the "complete sets of reductions" (introduced by D. Lankford). The whole correspondence discussed depends essentially on the fact that the cut rule in the Gentzen-system is eliminated. On the other hand the length of the proofs increases exponentially when the cut is eliminated. The analogue for the resolution is also true (Tseitin). Because cuts can be regarded as a kind of heuristics, this gives a logical motivation for the use of heuristics.

J.A.ROBINSON: Fast Unification

I talked about some specific implementation techniques and outlined the argument showing the running time to be essentially O(n) where n is the total number of distinct subexpressions in the expressions to be unified. The main idea is an efficient representation of these n expressions $E_1,\ldots,E_n$ by means of three boolean arrays variable [i], constant [i], list [i] and an array constituent [i], i=1,...,n, whose value is the list of integers $(i_1,\ldots,i_k)$ when E_i is the list $(E_{i_1},\ldots,E_{i_k})$. These arrays are "read-only" data. The working arrays are parent [i], representative [i], and weight [i], which have the meaning that $E_j \sim E_k$ iff class(j)=class(k), where we define recursively: class(n) = if parent[n] =n then n else class(parent[n]).
The expressions representative class(i) are maintained so that each class is represented by one of its nonvariable members, if such there be. The array weight counts the expressions rooted in i, in the graph (tree) structure defined by parent; and is used to maintain balanced trees. The consistency processing (cf. Huet's abstract) can then be done very rapidly by reference to the class representatives.

S.SICKEL: Clause Interconnectivity Graphs

A clause interconnectivity graph is a representation for sets of clauses that expresses the search for a proof as a graph searching problem. The search technique unrolls the graph into sets of trees. The trees grow in a well-defined breadth-first way that defines a measure of proof complexity.
All propositional calculus problems are solved at complexity level 0. In the general case the trees are evaluated for consistency of substitutions between developing solutions to dependent subgoals. Consistency restrictions increase as the search progresses, so that the branching rate decreases as the depth of search increases. Proofs are partitioned into equivalence classes represented by "proof schemas". The search produces schemas rather than individual proofs.

S.TÄRNLUND: Predicate Logic Programming

A. Syntax: We use Kowalski's procedure notation, which is a clause form for resolution logic.
B. Expressiveness: 1)Representation: The interpretation of predicate logic terms as data structures, which are manipulated by procedures, gives programs for representation of fundamental algorithms as well as complicated scenes such as cube rotations, towers, etc. .
2)Parallel execution: In addition to the representation power, the terms also give rise to computational efficiency by parallel processing.
3)Dual programming: Kowalski has pointed out that there is no distinction between input and output in predicate logic programming. We give an example, Knuth's insert algorithm, where this feature gives a program which can be executed efficiently also when asking for the input given the output.
C. Computational basis: It is shown that the simple system binary Horn set is Turing computable complete.

M.TYSON: Theorem Proving in Program Verification

One area that semi-automatic theorem provers may become useful is in the area of program verification. Some of the problems in this area will be discussed. These are drawn from experiences with W.W.Bledsoe's interactive theorem prover in use with the XIVUS verification system at the University of Texas and at Information Sciences Institute, Marina Del Ray, California.

G.WINTERSTEIN: Second-Order Unification

The purpose of my talk is to give a simple unification algorithm for the monadic case of second-order unification (string unification). The question whether it is decidable if there exists an unifier for two strings is shown to be equivalent with the existence of an upper bound for the application of the imitation rule. Several cases are pointed out where the existence of an unifier is seen immediately and the corresponding bound is a linear function of the length of the two strings.
Remark: However it has been pointed out (G.Huet) during the meeting that there exist pairs of strings which do not possess a linear bound and thus leaving the decidability for string unification still an open question.

C.Schippang, Aachen