nal Symposium on Programming. Proceedings, Girault and M. Paul. VI, 262 pages. 1984.

nd Tools for Computer Integrated Manufacturing. nn and U. Rembold. XVI, 528 pages. 1984.

e, Feedback Shift Registers. II, 1–2, 145 pages.

ternational Conference on Automated Deduction. stak. IV, 508 pages. 1984.

Machines: Decision Problems and Complexity. . Edited by E. Börger, G. Hasenjaeger and pages. 1984.

, Languages and Programming. Proceedings, aredaens. VIII, 527 pages. 1984.

s of Data Types. Proceedings, 1984. Edited by Queen and G. Plotkin. VI, 391 pages. 1984.

M 84. Proceedings, 1984. Edited by J. Fitch. 4.

, P-Functions and Boolean Matrix Factorization, 4.

ical Foundations of Computer Science 1984. Edited by M.P. Chytil and V. Koubek. XI, 581

ing Languages and Their Definition. Edited by 154 pages. 1984.

n Cognitive Ergonomics – Mind and Computers. Edited by G.C. van der Veer, M.J. Tauber, T.R.G. VI, 269 pages. 1984.

/ to Multiply Matrices Faster. XI, 212 pages. 1984.

are Tools Interfaces. Proceedings, 1983. Edited 164 pages. 1984.

ns of Software Technology and Theoretical Proceedings, 1984. Edited by M. Joseph and III, 468 pages. 1984.

2nd Annual Symposium on Theoretical Aspects e. Proceedings, 1985. Edited by K. Mehlhorn. 5.

h Project CIP. Volume I: The Wide Spectrum the CIP Language Group. XI, 275 pages. 1985.

Networks: An Advanced Course. Proceedings, lutchison, J. Mariani and D. Shepherd. VIII, 497

al Foundations of Software Development. Pro- lume 1: Colloquium on Trees in Algebra and '85). Edited by H. Ehrig, C. Floyd, M. Nivat and pages. 1985.

hods and Software Development. Proceedings, oquium on Software Engineering (CSE). Edited M. Nivat and J. Thatcher. XIV, 455 pages. 1985.

haghi, Time Series Package (TSPACK). III,

in Petri Nets 1984. Edited by G. Rozenberg of H. Genrich and G. Roucairol. VII, 467 pages.

an, Paragon. XI, 376 pages. 1985.

, J.P. Ansart, G. Hommel, L. Lamport, B. Liskov, . Schneider, Distributed Systems. Edited by gert. VI, 573 pages. 1985.

A Survey of Verification Techniques for Parallel ges. 1985.

# Lecture Notes in Computer Science

Edited by G. Goos and J. Hartmanis

230

## 8th International Conference on Automated Deduction

Oxford, England, July 27–August 1, 1986
Proceedings

Edited by Jörg H. Siekmann

Springer-Verlag
Berlin Heidelberg New York London Paris Tokyo

# THINKER

Francis Jeffry Pelletier
Departments of Philosophy and Computing Science
University of Alberta
Edmonton, Alberta T6G 2E1
Canada

THINKER is an automatic theorem proving system which generates proofs of theorems (including arguments with premises) in the natural deduction format of Kalish & Montague (1964). The logic in which THINKER operates is first-order predicate logic with identity (this is an update from the report, Pelletier 1982). The theorem to be proved, and any premises for the argument, are entered in a fully-parenthesized form, and can have any wff composed of & (and), v (or), > (if-then), ~ (not), = (if and only if), A (universal quantifer) E (existential quantifier), - (identity), variables, constants, and predicates of any adicity. THINKER performs no pre-processing of formulas (eg., into clause form) but instead operates directly on the natural form.

The natural deduction system allows for assumptions to be made at various places, and subproofs to be generated from these assumptions. The proof which is generated is fully formatted (indented, with subproofs displayed) and fully annotated (each line of the proof mentions which previous line it comes from). Handling of the subproofs is the trickiest part of the program, since once a subproof is complete, the lines generated in the course of constructing the subproof are no longer available for use in the rest of the proof.

The user interface for THINKER includes interactive prompting commands, various debugging facilities, a statistics collector (to see, for example, how many times certain ultimately unsuccessful strategies were tried), a "unfinished proof" examination mechanism, and a help facility that allows the user to add new lines to a partial proof or to suggest new subproofs to try. A post-processor is available to print out very tidy proofs on the Xerox 9700 printer.

THINKER has performed quite well on a wide variety of elementary problems (see Pelletier 1986 for a sample). Its success is mostly due to its use of a natural deduction format (which allows one to partition antecedent lines into different types, depending upon what its main connective is) and a heavy use of memory to "remember" (by means of hash tables) all formulas of all types that have already occurred in the proof. Thus, for a trivial example, if the current subgoal is the formula P, a check will be made to see whether (among other things) a formula (Q > P) occurs among the antecedent lines, and if so whether also the formula Q does. THINKER does this simply by storing the string (@ > P) in a hash table (if any formula of that form is an antecedent line). A pointer from this template tells what the actual values of @ these are - that is, for each formula of that form that is among the antecedent lines, what is the left side of the > . Since this too will be a string, it can be quickly looked up to determine whether it occurs in the antecedent lines. Templates are used for all connectives and free variables. Identity is handled by keeping track of equivalence classes of variables. (All these continually change due to new antecedent lines and completion of subproofs).

The reason that the management of formulas is so efficient has to do with the use of SPITBOL as the programming language. One merely needs to define each check as a SPITBOL pattern and use the SPITBOL TABLE facility. The use of strings, rather than lists or trees,

enables us to keep the advantages of the natural form (the advantages in selecting a proof strategy) while allowing fast access to the "form" of various formulas (since we need not process any tricky data structures).

THINKER is currently available on VAX 780/UNIX and AMDAHL 5680/MTS. Future extensions include the introduction of arbitrary functions, the use of (pseudo) parallelism to generate subproofs, and addiction of the lambda-calculus. This last is important since the ultimate goal of the system is to be attached to a natural language system to compute inferences and presuppositions. The NLU system under development makes heavy use of the lambda calculus in its representation of the logical form.

Pelletier, F.J. (1982) Completely Non-Clausal, Completely Heuristically Driven, Automatic Theorem Proving. Dept. Computing Science, University of Alberta, Tech. Report TR82-7.

Pelletier, F.J. (1986) "Seventy-Five Problems for Testing Automatic Theorem Provers" Jour, Automated Reasoning (forthcoming).