Technical Report CS75027-R

Automated Theorem-proving and Program Verification
An Annotated Bibliography

Deborah Y. Macock

November 1975

Virginia Polytechnic Institute and

State University

Blacksburg, Virginia 24061

The work represented here was supported by NSF Grant DCR74-18108.

This bibliography contains a synopsis of each reference, taken from one of three sources:

(1) the author's abstract;

(2) the review of the reference in ACM Computing Reviews;

(3) an abstract prepared locally.

The third alternative was chosen only if the first two were unavailable.

Each reference is identified by a number with prefix PV or A (program verification or automated theorem-proving) depending upon its contents, and the identification number is used for cross-referencing the entries. If a reference deals with both areas, it has two identification numbers. Also included is an author index, which is arranged alphabetically according to the first author's name if the source is co-authored. It is anticipated that this bibliography will be updated as research in the field progresses.

CR categories: 5.21, 5.24, 5.27, 4.6, 3.60

Key Words: automated theorem-proving, program correctness, program verification, software reliability, software evaluation

Author Index

Aiello, L., Aiello, M., and Weyhrauch, R. W., 'The Semantics of PASCAL in LCF', Stanford Artificial Intelligence Laboratory, Report No. STAN-CS-74-447, 1974. (PV41)

Allen, John, 'Preliminary User's Manual for an Interactive Theorem Prover', Stanford A. I. Lab.; Operating Note 73, 1973. (A44)

Anderson, Robert and Bledsoe, W. W., 'A Linear Format for Resolution with Merging and a New Technique for Establishing Completeness', JACM 17: 525-534, 1970. (A20)

Andrews, Peter B.,'Refutations by Matings', Dept. of Mathematics, Carnegie-Mellon, 1975. (A42)

Basu, Sanat K. and Misra, Jayadev, 'Proving Properties of Parallel Programs: An Axiomatic Approach', IEEE Trans. Software Engineering Vol. SE-1,1: 76-86, 1975. (PV33)

Birman, A., 'On Proving Correctness of Microprograms', IBM J. Research and Dev. 18,3: 250-266, 1974. (PV36)

Bledsoe, W. W., Boyer, R. S., and Hennemen, W. H., 'Computer Proofs of Limit Theorems', Artificial Intelligence 3: 27-60, 1972. (A29)

Bledsoe, W. W. and Bruell, Peter, 'A Man-Machine Theorem-Proving System', Artificial Intelligence 5: 51-72, 1974. (A12)

Bledsoe, W. W. and Tyson, Mabry, 'Typing and Proof by Cases in Program Verification', U.T. Math. Dept. Memo ATP 15, 1975. (A31)

Bledsoe, W. W. and Tyson, Mabry, 'The UT Interactive Prover', U.T. Math. Dept. Memo ATP 17, 1975. (A30)

Boyer, Robert S. and Moore, J. Strother, 'Proving Theorems About LISP Functions', JACM 22: 129-144, 1975. (A8/PV2)

Bundy, Alan, 'Doing Arithmetic with Diagrams', Third IJCAI: 130-138, 1973. (A37)

Burstall, R. M., 'Proving properties of programs by structural induction', Computer Journal 12,1: 41-48, 1969. (PV7)

Chang, C. L., 'The Unit Proof and the Input Proof in Theorem Proving', JACM 17: 698-707, 1970. (A21)

Chang, C. L. and Lee, R. C. T., Symbolic Logic and Mechanical Theorem Proving, Academic Press, 1974. (A28/PV17)

Cooper, D. C., 'Programs for Mechanical Program Verification', Machine Intelligence 6: 43-59, 1971. (PV1)

Darlington, J. L.,'Automatic Theorem Proving with Equality Substitutions and Mathematical Induction', Machine Intelligence 3: 113-127, 1968. (A35)

Davis, Martin and Putnam, Hilary, 'A Computing Procedure for Quantification Theory', JACM 7: 201-215, 1960. (A36)

deBakker, J. W., 'Semantics of Programming Languages', Ad. in Info. Systems Science Vol 2: 173-227, 1969. (PV21)

Dijkstra, E. W., 'A Constructive Approach to the Problem of Program Correctness', BIT 8: 174-186, 1968. (PV9)

Elspas, B., Green, M. W., and Levitt, Karl, 'Software Reliability', Computer 4,1: 21-27, 1971. (PV23)

Elspas, B., Levitt, K. N., Waldinger, R. J., and Waksman, A., 'An Assessment of Techniques for Proving Program Correctness', ACM Computing Surveys 4,2: 97-147, 1972. (PV4)

Ernst, G. W. and Hookway, R. J., 'Formulating inductive assertions for program verification', Case Western Reserve Univ., Report 1165, 1975. (A15/PV14)

Fishman, D. H., 'A Problem-Oriented Search Procedure for Theorem Proving', COINS Tech. Rep. 75C-4 (U. Mass.), 1975. (A33)

Florentin, J. J., 'Language Definition and Compiler Validation', Machine Intelligence 3: 33-41, 1968. (PV22)

Floyd, Robert W., 'Assigning Meanings to Programs', Proc. Symp. in Applied Mathematics Vol. 19: Mathematical Aspects of Computer Science (AMS): 19-32, 1967. (PV18)

Floyd, Robert W., 'Toward Interactive Design of Correct Programs', IFIP 71: 7-10, 1971. (PV42)

Gelernter, H., 'Realization of a Geometry-Theorem Proving Machine', Computers and Thought, Feigenbaum, E. A. and Feldman, J., eds., McGraw-Hill Book Company, 1963. (A38)

Gerhart, Susan, 'On the Relationship Between Structured Programming and Correctness Proofs', ICASE Report 75-15, NASA Langley Research Center, 1975. (PV43)

German, Steven M. and Wegbreit, Ben, 'A Synthesizer of Inductive Assertions', IEEE Trans. Software Engineering Vol. SE-1,1: 68-75, 1975. (PV32)

Good, Donald I., 'Toward a Man-Machine System for Proving Program Correctness', Ph.D. Thesis, Comp. Science Dept., U. of Wisc., 1970. (PV25)

Good, Donald I., London, Ralph L., and Bledsoe, W. W., 'An Interactive Program Verification System', IEEE Trans. Software Engineering Vol. SE-1,1: 59-67, 1975. (PV31)

Habermann, A. N., 'The Correctness Proof of a Quadratic-Hash Algorithm', Dept. of Comp. Science, Carnegie-Mellon, 1975. (PV20)

v.Henke, Friedrich W. and Luckham, David C., 'Automatic Program Verification III: A Methodology for Verifying Programs', Stanford Artificial Intelligence Laboratory, Rep. No. STAN-CS-74-474, 1974. (PV3)

Hewitt, Carl E. and Smith, Brian, 'Towards a Programming Apprentice', IEEE Trans. Software Engineering Vol. SE-1,1: 26-45, 1975. (PV29)

Hoare, C. A. R., 'An Axiomatic Basis for Computer Programming', CACM 12,10: 576-583, 1969. (PV10)

Huber, Hartmut G. M. and Morris, Alfred H., 'Resolution in First Order Theories', NWL Tech. Report No. TR-2405, 1970. (A5)

Huber, Hartmut G. M. and Morris, Alfred H., 'Primary Paramodulation', NWL Tech. Report No. TR-2552, 1971. (A6)

Huber, Hartmut G. M. and Morris, Alfred H., 'Contracted Resolution', NWL Tech. Report No. TR-2655, 1972. (A7)

Igarashi, S., London, R. L., and Luckham, D. C., 'Automatic Program Verification I: A Logical Basis and its Implementation', Stanford Artificial Intelligence Laboratory Report No. STAN-CS-73-365, 1973. (PV40)

Kang, Andy N. C. and Wang, Shih-Ho, 'An Extension of the Direct Method for Verifying Programs', Tech. Rep. CS75022-R, Dept. of Computer Science, Virginia Polytechnic Institute and State University, 1975. (PV44)

Katz, Shmuel M. and Manna, Zohar, 'A Heuristic Approach to Program Verification', Third IJCAI: 500-512, 1973. (PV5)

King, J. C., 'A Program Verifier', IFIP 71: 234-249, 1972. (PV24)

King, James C. and Floyd, Robert W., 'An Interpretation-Oriented Theorem Prover over Integers', J. of Computer and System Sciences 6: 305-323, 1972. (A47)

Kowalski, R. and Hayes, P. J., 'Semantic Trees in Automatic Theorem Proving', Machine Intelligence 4: 87-101, 1969. (A4)

Kowalski, Robert and Kuehner, Donald, 'Linear Resolution with Selection Function', Artificial Intelligence 2: 227-260, 1971. (A22)

Lanzarone, G. A., 'Proof of Program Correctness: A Review', Honeywell Computer Journal 6,1: 38-42, 1972. (PV6)

Lauer, P., 'Consistent Formal Theories of the Semantics of Programming Languages', IBM Laboratory Vienna TR 25.121, 1971. (PV35)

Leeman, George B., Jr., 'Some Problems in Certifying Microprograms', IEEE Trans. Comp. Vol. C-24,5: 545-553, 1975. (PV38)

Leeman, George B., Jr., Carter, William C., and Birman, Alexander, 'Some Techniques for Microprogram Validation', IFIP 74: 76-80, 1974. (PV37)

Liskov, B. H. and Zilles, S., 'Specification Techniques for Data Abstractions', IEEE Trans. Software Engineering Vol. SE-1,1: 7-19,

1975. (PV27)

London, R. L., 'Bibliography on proving the correctness of computer programs', Machine Intelligence 5: 569-580, 1970. (PV12)

London, R. L., 'Proving Programs Correct: Some Techniques and Examples', BIT 10: 168-182, 1970. (PV11)

Loveland, D. W., 'A Unifying View of Some Linear Herbrand Procedures', JACM 19: 366-384, 1972. (A19)

Luckham, David, 'Some Tree-paring Strategies for Theorem Proving', Machine Intelligence 3: 95-112, 1968. (A3)

Manna, Zohar, Mathematical Theory of Computation, McGraw-Hill Book Company, 1974. (A14/PV13)

Manna, Zohar, 'The Correctness of Programs', Journal of Computer and System Sciences 3: 119-127, 1969. (PV8)

Manna, Zohar and Waldinger, Richard J., 'Toward Automatic Program Synthesis', CACM 14,3: 151-165, 1971. (A34)

McCarthy, J., 'Towards a mathematical science of computation', IFIP 62: 21-28, 1962. (A16/PV15)

McCarthy, J. and Painter, J. A., 'Correctness of a compiler for arithmetic expressions', Proc. Symp. in Applied Math., Vol. 19: Math. Aspects of Computer Science (AMS): 33-41, 1967. (PV16)

McKeeman, W. M., 'On Preventing Programming Languages from Interfering with Programming', IEEE Trans. Software Engineering Vol. SE-1,1: 19-26, 1975. (PV28)

Meltzer, B., 'Theorem-proving for Computers: Some results on resolution and renaming', Computer Journal 8: 341-343, 1966. (A11)

Morgan, Charles G., 'Methods for Automated Theorem Proving in Non-Classical Logics', Dept. of Philosophy, Univ. of Victoria, 1975. (A41)

Naur, Peter, 'Proof of algorithms by general snapshots', BIT 6: 310-316, 1966. (PV19)

Owicki, Susan and Gries, David,'Proving Properties of Parallel Programs: An Axiomatic Approach', Dept. of Comp. Science, Cornell, TR 75-243, 1975. (PV34)

Ramamoorthy, C. V. and Ho, S. B. F., 'Testing Large Software with Automated Software Evaluation Systems', IEEE Trans. Software Engineering Vol. SE-1,1: 46-58, 1975. (PV30)

Robinson, G. and Wos, L., 'Paramodulation and Theorem-proving in First-Order Theories with Equality', Machine Intelligence 4: 135-150, 1969. (A26)

Robinson, J. A., 'A Machine-Oriented Logic Based on the Resolution Principle', JACM 12: 23-41, 1965. (A1)

Robinson, J. A., 'A Note on Mechanizing Higher Order Logic', Machine Intelligence 5: 123-133, 1970. (A49)

Robinson, J. A.,'A review of automatic theorem proving', Proc. Symp. in Applied Math. (AMS): 1-18, 1967. (A18)

Robinson, J. A., 'An Overview of Mechanical Theorem Proving', Theoretical Approaches to Non-Numerical Problem Solving: 2-20, Springer-Verlag, 1970. (A48)

Robinson, J. A., 'Automatic Deduction with Hyper-Resolution', Int. J. of Computer Math. 1: 227-234, 1965. (A24)

Robinson, J. A., 'Heuristic and Complete Processes in the Mechanization of Theorem Proving', Systems and Computer Science, Univ. of Ontario Press, 1967. (A46)

Robinson, J. A., 'New Directions in Mechanical Theorem Proving', IFIP 68: 63-67, 1968. (A43)

Robinson, J. A., 'The generalized resolution principle', Machine Intelligence 3: 77-93, 1968. (A45)

Robinson, J. A., 'Theorem-Proving on the Computer', JACM 10: 163-174, 1963. (A10)

Sickel, Sharon, 'Variable Range Restrictions in Resolution Theorem Proving', U.C. Santa Cruz, 1975. (A32)

Slagle, James R., 'Automatic Theorem Proving with Built-in Theories Including Equality, Partial Ordering, and Sets', JACM 19: 120-135, 1972. (A25)

Slagle, James R., 'Automatic Theorem Proving with Renamable and Semantic Resolution', JACM 14: 687-697, 1967. (A2)

Stickel, M. E., 'A Complete Unification Algorithm for Associative-Commutative Functions', Dept. of Computer Science, Carnegie-Mellon, 1975. (A39)

Stickel, Mark E., 'The Programmable Strategy Theorem Prover: An Implementation of the Linear MESON Procedure', Dept. of Computer Science, Carnegie-Mellon, 1974. (A40)

Vere, Steven A., 'Induction of concepts in the predicate calculus', Dept. of Information Engineering, U. of Illinois at Chicago Circle, 1975. (A27)

Waldinger, R. J., and Levitt, K. N., 'Reasoning about Programs', Artificial Intelligence 5: 235-316, 1974. (PV26)

Wang, Hao, 'Formalization and Automatic Theorem-proving', IFIP 65: 51-58, 1965. (A13)

Wang, Hao, 'Toward Mechanical Mathematics', IBM J. Res. and Dev. 4: 2-22, Jan., 1960. (A17)

Wegbreit, Ben, 'Heuristic Methods for Mechanically Deriving Inductive Assertions', Third IJCAI: 524-536, 1973. (PV39)

Wos, L., Carson, D., and Robinson, G., 'The unit preference strategy in theorem proving', Prcc. AFIPS 1964 FJCC: 615-621, 1964. (A23)

Wos, L., Robinson, G., and Carson, D., Efficiency and Completeness of the Set of Support Strategy in Theorem Proving', JACM 12: 536-541, 1965. (A9)

Robinson, J. A., 1965

A Machine-Oriented Logic Based on the Resolution Principle

JACM 12:23-41, 1965

Ref: A10, A36

Abstract

Theorem-proving on the computer, using procedures based on the fundamental theorem of Herbrand concerning the first-order predicate calculus, is examined with a view towards improving the efficiency and widening the range of practical applicability of these procedures. A close analysis of the process of substitution (of terms for variables), and the process of truth-functional analysis of the results of such substitutions, reveals that both processes can be combined into a single new process (called resolution), iterating which is vastly more efficient than the older cyclic procedures consisting of substitution stages alternating with truth-functional analysis stages.

The theory of the resolution process is presented in the form of a system of first-order logic with just one inference principle (the resolution principle). The completeness of the system is proved; the simplest proof-procedure based on the system is then the direct implementation of the proof of completeness. However, this procedure is quite inefficient, and the paper concludes with a discussion of several principles (called search principles) which are applicable to the design of efficient proof-procedures employing resolution as the basic logical process.

Slagle, James R., 1967

Automatic Theorem Proving With Renamable and Semantic Resolution

JACM 14:687-697, 1967

Ref: A1, A9, A11, A13, A18, A23, A24, A36, A38

Abstract

The theory of J. A. Robinson's resolution principle, an inference rule for first-order predicate calculus, is unified and extended. A theorem-proving computer program based on the new theory is proposed and the proposed semantic resolution program is compared with hyper-resolution and set-of-support resolution programs. Renamable and semantic resolution are defined and shown to be identical. Given a model M, semantic resolution is the resolution of a latent clash in which each "electron" is at least sometimes false under M; the nucleus is at least sometimes true under M.

The completeness theorem for semantic resolution and all previous completeness theorems for resolution (including ordinary, hyper-, and set-of-support resolution) can be derived from a slightly more general form of the following theorem. If U is a finite, truth-functionally unsatisfiable set of nonempty clauses and if M is a ground model, then there exists an unresolved maximal semantic clash $\{E_1, E_2, \ldots, E_q, C\}$ with nucleus C such that any set containing C and one or more of the electrons $E_1, E_2, \ldots, E_q$ is an unresolved semantic clash in U.

Luckham, David, 1968

Some Tree-paring Strategies for Theorem Proving

Machine Intelligence Vol. 3: 95-112

Ref: A1, A9, A11, A13, A23, A24

Abstract

Proofs of theorems from elementary algebra and number theory have been obtained using a resolution program with a memory capacity limited to 200 clauses. The object has been to study the effectiveness of model partition strategies in cutting down the number of deductions considered. For this reason proofs were obtained without the unit preference strategy where possible. Our results indicate that these strategies are certainly useful but often have the unfortunate consequence of excluding the shortest proof from consideration. The question of which is the best such strategy for a given problem is not straightforward.

Kowalski, R. and Hayes, P. J., 1969

Semantic Trees in Automatic Theorem Proving

Machine Intelligence 4:87-101

Ref: A1, A2, A3, A9, A11, A18, A24, A45

Review (R. Anderson, Austin, Texas)

This paper presents a very substantial application (and clarification) of the concept of semantic trees, as originally introduced in J. A. Robinson's paper, "The Generalized Resolution Principle," [*Machine Intelligence* 3, pp. 77-93]. The concept of semantic trees provides a very powerful and elegant framework for viewing the completeness proofs of most of the well-known resolution-based deductive systems for first-order logic, as utilized in automatic theorem proving. The authors proceed to demonstrate the power of this concept by proving, in a most clear and unified manner, the completeness of clash resolution, binary resolution, *A-M*-clash resolution and hyper-resolution. The main new results contained in this paper are that several improvements in the above deductive systems can be made with regard to the generation of factors and the notion of A-ordering. Utilizing these improvements, a very attractive modification of P_1-deduction is presented.

Huber, Hartmut G. M. and Morris, Alfred H., 1970

Resolution in First Order Theories

NWL Tech. Report No. TR-2405

Ref: A1, A2, A3, A4, A9, A18, A36, A45

Abstract

In this paper the basic principles that are involved in resolution theory for first order theories are examined. Resolution theory is developed from an axiomatic point of view, which provides a general foundation for specific proof finding strategies in first order theories. Most of the specific strategies that have been formulated for the predicate calculus are presented as natural applications of this theory.

Huber, Hartmut G. M. and Morris, Alfred H., 1971

Primary Paramodulation

NWL Tech. Report No. TR-2552

Ref: A2, A5, A20, A26

Abstract

Computers can directly use mathematical logic, that is first-order predicate calculus, in decision making and problem solving by the application of proof strategies involving resolution and paramodulation. A new category of problem solving strategies is defined which has the interesting and unusual property that refutations are formed by mixing certain primary paramodulants with resolvents obtained from any complete strategy in the predicate calculus without equality. Also, a new and easily applied technique is given for verifying that subsumed clauses can be eliminated from deductions. This technique is applied to all the strategies considered in this paper.

Huber, Hartmut G. M. and Morris, Alfred H., 1972

Contracted Resolution

NWL Tech. Report No. TR-2655

Ref: A6, A20

Abstract

In order to improve the capability of resolution programs for automatic decision making, a new rule for inference called contracted resolution is proposed. Under this rule of inference more literals are often eliminated than in a pairwise Robinson resolvent. Contracted resolution is shown to be sound and complete, requiring only refutations that are input deductions in the Chang and Slagle sense. Subsumption is defined in this environment, and it is shown that subsumed clauses can be deleted.

Boyer, Robert S. and Moore, J. Strother, 1975

Proving Theorems About LISP Functions

JACM 22: 129-144, 1975

Ref: A29

PV1, PV5, PV7, PV16, PV18, PV19, PV25, PV40

Abstract

Program verification is the idea that properties of programs can be precisely stated and proved in the mathematical sense. In this paper, some simple heuristics combining evaluation and mathematical induction are described, which the authors have implemented in a program that automatically proves a wide variety of theorems about recursive LISP functions. The method the program uses to generate induction formulas is described at length. The theorems proved by the program include that REVERSE is its own inverse and that a particular SORT program is correct. A list of theorems proved by the program is given.

Wos, L., Robinson, G., and Carson, D., 1965

Efficiency and Completeness of the Set of Support Strategy in Theorem Proving

JACM 12: 536-541, 1965

Ref: A1, A23

Abstract

One of the major problems in mechanical theorem proving is the generation of a plethora of redundant and irrelevant information. To use computers effectively for obtaining proofs, it is necessary to find strategies which will materially impede the generation of irrelevant inferences. One strategy which achieves this end is the set of support strategy. With any such strategy two questions of primary interest are that of its efficiency and that of its logical completeness. Evidence of the efficiency of this strategy is presented, and a theorem giving sufficient conditions for its logical completeness is proved.

Robinson, J. A., 1963

Theorem-Proving on the Computer

JACM 10:163-174, 1963

Ref: A17, A36

Review (Donald Monk, Berkeley, California)

This article is based on, and presupposes knowledge of, a paper by M. Davis and H. Putnam [*J. ACM* 7 (1960), 201-215]; reference is also made to a mimeographed paper by M. Davis, which is unfortunately not available to the reviewer. The Davis-Putnam method may be summarized thus: interested in whether a pure first-order formula A is refutable, transform A into a universal formula $\mathcal{B}$ by introducing Skolem functions; A is not refutable if all open formulas obtainable from A by substituting variable-free terms (VFT) are sententially consistent. The authors then describe two methods: (1) for generating n-tuples of VFT; (2) for testing sentential consistency of open formulas $\mathcal{B}$ (assumed to be in conjunctive normal form).

The present paper describes results of a 704 program for this method. Two modifications are made: a set of n-tuples of VFT is introduced explicitly for each problem, so that (1) above is not mechanized (this seems quite reasonable; the method suggested by Davis and Putnam for (1), lexicographic ordering, is unwieldy and was not even used by them). Second, some time-saving modifications of method (2) are made. The results described range over several parts of elementary mathematics; e.g., the theorem "if α is prime then $\sqrt{\alpha}$ is irrational" was proved by the machine. The machine ran out of time trying to prove that there do not exist two orthogonal Latin squares of order 6. The paper is not written very precisely. For example, the definition of q on page 164 differs from its use on page 165. One is surprised to find on page 166 a predicate $P(x,y,z)$ expressing that $z = x \cdot y$; a binary operation symbol $\cdot$ is more natural--operation symbols are allowed to be primitive, as well as being introduced as Skolem functions.

Meltzer, B., 1966

Theorem-proving for Computers: Some results on resolution and renaming

Computer Journal 8: 341-343, 1966

Ref: A1

Abstract

It is shown that J. A. Robinson's P_1-deduction is a special case of a large class of types of deduction by resolution, an optimum choice from which should be possible for any particular theorem to be proved. Some further results, based on the operation of renaming literals by means of their negations, are obtained and suggest an alternative approach to automatic deduction.

A12

Bledsoe, W. W., and Bruell, Peter, 1974

A Man-Machine Theorem-Proving System

<u>Artificial Intelligence</u> 5: 51-72, 1974

Ref: A25, A29

Abstract:

This paper describes a man-machine theorem-proving system at the University of Texas at Austin which has been used to prove a few theorems in general topology. The theorem (or subgoal) being proved is presented on the scope in its natural form so that the user can easily comprehend it and, by a series of interactive commands, can help with the proof when he desires. A feature called DETAIL is employed which allows the human to interact only when needed and only to the extent necessary for the proof.

The program is built around a modified form of IMPLY, a natural-deduction-like theorem proving technique which has been described earlier. A few examples of proofs are given.

A13

Wang, Hao, 1965

Formalization and Automatic Theorem-proving

<u>Proc. IFIP Congress 65</u>: 51-58

Ref: A1, A10, A17, A23, A36

Local

The main purpose of the paper is to give some new examples of mechanizing proofs in number theory and quantification theory. A summary of existing results in the area of mechanical mathematics is given.

A14/PV13

Manna, Zohar, 1974

Mathematical Theory of Computation

McGraw-Hill Book Company, 1974

Ref: A1, A8/PV2, A16/PV15, A17, A26, A28/PV17, A34, A36, A43
PV1, PV5, PV7, PV8, PV9, PV10, PV11, PV16, PV18, PV19, PV24
PV25, PV26, PV39, PV40, PV42

Local

In this book, the author attempts to treat both the practical and the theoretical aspects of the mathematical theory of computation. The topics discussed include Robinson's resolution, the verification of both flowchart and ALGOL-like programs, flowchart schemas, and the fixpoint theory of programs.

A15/PV14

Ernst, G. W. and Hookway, R. J., 1975

Formulating inductive assertions for program verification

Case Western Reserve University Report #1165

Ref: A12
PV16, PV18, PV26, PV31, PV40

Abstract

This research studies the interface between program verification and mechanical theorem proving. Both of these areas give rise to constraints that affected the design of the function definition facility, described in this report, that is used in the CWRU program verification system. The functions that occur in inductive assertions often have recursive definitions that may contain higher order variables. However, both recursive definitions and higher order variables are difficult to process mechanically. Our function definition facility is a compromise between these design constraints.

This report also describes a method for stating inductive assertions in such a way as to simplify proving them mechanically. The basic idea is to replace existentially quantified variables, which may be function variables, by values stored in various program variables. Simple examples of how this effects both the formulation of assertions and their proof is given. We also apply this technique to a relatively complex example which involves some list processing.

McCarthy, J., 1962

Towards a Mathematical Science of Computation

<u>IFIP</u> 62: 21-28, 1962

Ref:

Local

This paper constitutes one of the earliest considerations of the feasibility of automated theorem-proving and the verification of programs. A method for proving statements about recursive functions is presented which involves describing the functions in terms of conditional expressions and implementing the principle of "recursion induction". The concepts of state vectors and abstract syntax are introduced, and their roles in the definition of the semantics of a programming language are discussed.

Wang, Hao, 1960

Toward Mechanical Mathematics

<u>IBM J. Res. and Dev.</u> 4:2-22, Jan., 1960

Ref:

Abstract

Results are reported here of a rather successful attempt at proving all theorems, totalling near 400, of <u>Principia Mathematica</u> which are strictly in the realm of logic, viz., the restricted predicate calculus with equality. A number of other problems of the same type are discussed. It is suggested that the time is ripe for a new branch of applied logic which may be called "inferential" analysis, which treats proofs as numerical analysis does calculations. This discipline seems capable, in the not too remote future, of leading to machine proofs of difficult new theorems. An easier preparatory task is to use machines to formalize proofs of known theorems. This line of work may also lead to mechanical checks of new mathematical results comparable to the debugging of a program.

Robinson, J. A., 1967

A review of automatic theorem proving

Proc. Symp. in Applied Math. Vol. 19 (AMS): 1-18

Ref: A1, A9, A10, A11, A13, A17, A23, A24, A36

Review (F. Baskett III and W. W. Bledsoe, Austin, Texas)

This article is a good review of the work done in automatic theorem-proving up to 1967 and provides a background for much of the work being done today. It is recommended to anyone being introduced to the subject. It introduces the basic formalism of first-order, quantifier-free logic and the notions of validity and satisfiability in terms of this logic. It presents the Herbrand universe, the idea of instances and interpretations in that universe, and a proof of the fundamental theorem of logic. The unification algorithm, which lies at the heart of automatic theorem-proving in first-order logic, is clearly presented, and the important theorems about unifiers are stated.

After giving a very brief treatment of the type of theorem-proving procedures proposed by Prawitz, Davis, Chinlund, and Loveland, the author turns to "clashes and their resolution." Clash resolution, which is first introduced in this paper, is a generalization of previous types of resolution procedures. A thorough completeness proof is given and several operational refinements are suggested. The clarity of the proof is remarkable and almost unique in this field. Although clash resolution is still an open topic, subject to more sophisticated development, it is now somewhat historical.

With regard to what has been done since this review paper was written, it should be noted that the author has now replaced the interpretation tree and the proof tree used in this article by a single, unified "semantic tree" in which the equality relation is built-in. Indeed, much of the current

research is concerned with building equality into the logic in an efficient manner. Many, including Prawitz and the author are now investigating automatic proof procedures in higher order logics based on Henkin completeness of the theory of types.

Loveland, D. W., 1972

A Unifying View of Some Linear Herbrand Procedures

JACM 19: 366-384, 1972

Ref: A1, A2, A20, A21, A22, A23, A36

Abstract

The linked conjunct, resolution, matrix reduction, and model elimination proof procedures constitute a nearly exhaustive list of the basic Herbrand proof procedures introduced in the 1960's. Each was introduced as a hopefully efficient complete procedure for the first order predicate calculus for the purpose of mechanical theorem proving. This paper contains a demonstration that versions of these procedures can be highly related in their design. S-linear resolution, a particular strategy of resolution previously proposed, is seen to possess a natural refinement isomorphic at ground level to a refinement of the model elimination procedure. There is also an isomorphism at the general level between a less natural S-linear resolution refinement and the model elimination refinement. The model elimination procedure is also interpreted within the linked conjunct and matrix reduction procedures. An alternate interpretation of these results is that, very roughly, the procedures other than resolution can be viewed as forms of linear resolution.

Anderson, Robert and Bledsoe, W. W., 1970

A Linear Format for Resolution With Merging and a New Technique for Establishing Completeness

JACM 17: 525-534, 1970

Ref: A1, A9, A18, A24, A26

Abstract

A new technique is given for establishing the completeness of resolution-based deductive systems for first-order logic (with or without equality) and several new completeness results are proved using this technique. The technique leads to very simple and clear completeness proofs and can be used to establish the completeness of most resolution-based deductive systems reported in the literature. The main new result obtained by means of this technique is that a linear format for resolution with merging and set of support and with several further restrictions is a complete deductive system for the first-order predicate calculus.

Chang, C. L., 1970

The Unit Proof and the Input Proof in Theorem Proving

<u>JACM</u> 17: 698-707, 1970

Ref: A1, A2, A3, A9, A11, A18, A23, A36, A45

Abstract

A resolution in which one of the two parent clauses is a unit clause is called a unit resolution, whereas a resolution in which one of the two parent clauses is an original input clause is called an input resolution. A unit (input) proof is a deduction of the empty clause □ such that every resolution in the deduction is a unit (input) resolution. It is proved in the paper that a set S of clauses containing its unit factors has a unit proof if and only if S has an input proof. A LISP program implementing unit resolution is described and results of experiments are given.

Kowalski, Robert and Kuehner, Donald, 1971

Linear Resolution with Selection Function

<u>Artificial Intelligence</u> 2: 227-260, 1971

Ref: A4, A9, A18, A20

Abstract

Linear resolution with selection function (SL-resolution) is a restricted form of linear resolution. The main restriction is effected by a selection function which chooses from each clause a single literal to be resolved upon in that clause. This and other restrictions are adopted to linear resolution from Loveland's model elimination.

We show that SL-resolution achieves a substantial reduction in the generation of redundant and irrelevant derivations and does so without significantly increasing the complexity of simplest proofs. We base our argument for the increased efficiency of SL-resolution upon precise calculation of these quantities.

A more far reaching advantage of SL-resolution is its suitability for heuristic search. In particular, classification trees, subgoals, lemmas, and/or search trees can all be used to increase the efficiency of finding refutations. These considerations alone suggest the superiority of SL-resolution to theorem-proving procedures constructed solely for their heuristic attraction.

From comparison with other theorem-proving methods, we conjecture that best proof procedures for first order logic will be obtained by further elaboration of SL-resolution.

A23

Wos, L., Carson, D., and Robinson, G., 1964

The unit preference strategy in theorem proving

Proc. AFIPS 1964 FJCC: 615-621

Ref: A1, A36

Review (Ralph L. London, Madison, Wisconsin)

This paper reports on two refinements made in the application of the resolution principle for the machine proof of theorems which are expressed in a conjunctive normal form with implicit quantification. These improvements take the form of reordering the generation of resolvents to produce, first, those which are shorter (unit preference strategy), and those which are derived from the join of the denial of the conclusion of the theorem under consideration and the special hypotheses of the theorem (set of support strategy). Excluded from the set of support are the basic axioms of the theory being studied.

The improved search algorithms are described, and their soundness and completeness are proved. As examples, three theorems from elementary group theory are presented together with four proofs generated by the computer. (Two proofs, using different sets of support, are given for one theorem.) To demonstrate the usefulness of the strategies, data are presented on unsuccessful attempts to prove two of the examples without the above strategies. Several other strategies are suggested, but no evaluation of them is presented.

A24

Robinson, J. A., 1965

Automatic Deduction with Hyper-Resolution

Int. J. Computer Math. 1:227-234, 1965

Ref: A1, A10, A23, A36

Local

Knowledge of Robinson's earlier work on the resolution principle (A1) is assumed in this paper. The concept of hyper-resolution is introduced as a means of eliminating much of the unnecessary computing in the original resolution procedure. The completeness of the system is proved, and two examples are given which illustrate the difference in efficiency between the hyper-resolution method and the original method.

Slagle, James R., 1972

Automatic Theorem Proving with Built-in Theories Including Equality,
Partial Ordering, and Sets

JACM 19:120-135, 1972

Ref: A1, A2, A4, A9, A13, A18, A23, A26, A36

Abstract

To make further progress, resolution principle programs need to make better inferences and to make them faster. This paper presents a fairly general approach for taking some advantages of the structure of special theories, for example, the theories of equality, partial ordering, and sets. The object of the approach is to replace some of the axioms of a given theory by (refutation) complete, valid, efficient (in time) inference rules. The author believes that the new rules are efficient because:

(1) inference for inference, a computer program embodying the rules should be faster than a resolution program computing with the corresponding axioms;

(2) more importantly, certain troublesome inference made by resolution are avoided by the new rules.

In this paper, the three main applications of the approach concern "building-in" the theories of equality, partial ordering, and sets and may be stated roughly as follows:

(1) If only $\{x=x\}$ is retained from the equality axioms, and if the others are replaced by the functionally reflexive axioms and the rule of renamable paramodulation, refutation completeness is preserved.

(2) If only $\{x \subseteq x\}$ is retained from the eight (not all independent) partial ordering axioms for $\{=, \subseteq, \subset\}$ and if the other seven are replaced by the rules $r(\subseteq, \subset)$ and $r(\subset)$, refutation completeness is preserved.

(3) If a certain seven of the twenty-four set axioms are retained and if the remaining seventeen are replaced by the rules $r(\subseteq, \subseteq)$, $r(\subset)$, $r(\in)$, $r(c)$(complement), $r(\cup)$, $r(\cap)$, and $r(u)$ (unit sets), refutation completeness is preserved.

Robinson, G. and Wos, L., 1969

Paramodulation and Theorem-proving in First-Order
Theories with Equality

Machine Intelligence 4: 135-150

Ref: A1, A2, A9, A23, A35, A36

Review (R. Anderson and W. W. Bledsoe, Austin, Texas)

This paper presents a method in automatic theorem proving, called paramodulation, for treating the equality relation in the first-order predicate calculus with equality. It is specifically designed to operate in conjunction with the rule of inference known as resolution, and together with resolution constitutes a complete deductive system for first-order logic with equality. Paramodulation is essentially a substitution rule for equals tailored to the specific logical format (clausal form) used in resolution-based deductive systems. The authors compare paramodulation with another method for treating equality, that of adding an appropriate set of axioms for equality, and claim for paramodulation several advantages in terms of proof brevity, naturalness, immediacy, convergence, and generality.

While it seems clear that paramodulation is one of the most effective methods proposed so far for handling equality, it is the reviewer's experience that any treatment of equality adds several orders of magnitude to the rate of growth of debris during the proof search, and that considerable ingenuity will be required to obtain enough efficiency for a practical system. In this regard it should be mentioned that paramodulation has now been shown compatible with the set of support strategy and hyper-resolution.

A27

Vere, Steven A., 1975

Induction of concepts in the predicate calculus

Dept. of Information Engineering, U. of Illinois at Chicago Circle

Ref:

Abstract

Positive and negative instances of a concept are assumed to be described by a conjunction of literals in the predicate calculus, with terms limited to constants and universally quantified variables. A graph representation of a conjunction of literals, called a "product graph" is introduced. It is desirable to merge positive instances by generalization, while maintaining discrimination against negative instances. This is accomplished by an induction procedure which operates on the product graph form of these positive and negative instances. The correctness of the procedure is proven, together with several related results of direct practical significance. This work is directed to the goal of providing a formal model for the inductive processes which are observed in artificial intelligence studies in specialized areas.

A28/PV17

Chang, C. L. and Lee, R. C. T., 1973

Symbolic Logic and Mechanical Theorem Proving

Academic Press, 1973

Ref: A1, A2, A3, A4, A6, A9, A10, A11, A13, A16/PV15, A17, A18, A19, A20, A21, A22, A23, A24, A25, A26, A29, A34, A35, A36, A38, A43, A45, A47, A49

PV1, PV7, PV8, PV9, PV10, PV12, PV18, PV19, PV22, PV25, PV42

Local

This book provides an excellent introduction to symbolic logic in addition to an extensive discussion of mechanical theorem proving and such applications as program analysis, program synthesis, question answering, and problem solving. Several different techniques are presented and compared (including Robinson's resolution principle, linear and semantic resolution, and paramodulation), and numerous examples are given. In addition to a bibliography at the end of the text, references are included after each section.

Bledsoe, W. W., Boyer, R. S., and Henneman, W. H., 1972

Computer Proofs of Limit Theorems

Artificial Intelligence 3:27-60, 1972

Ref: A1, A17

Abstract

Some relatively simple concepts have been developed which, when incorporated into existing automatic theorem proving programs (including those using resolution), enable them to prove efficiently a number of the limit theorems of elementary calculus, including the theorem that differentiable functions are continuous. These concepts include: (1) A limited theory of types, to designate whether a given variable belongs to a certain interval on the real line, (2) An algebraic simplification routine, (3) A routine for solving linear inequalities, applicable to all areas of analysis, and (4) A "limit heuristic", designed especially for the limit theorems of calculus.

Bledsoe, W. W., and Tyson, Mabry, 1975

The UT Interactive Prover

UT Automatic Theorem Proving Project #17

Ref: A8/PV2, A17, A31, A35, A37, A38
PV26

Abstract

The interactive theorem prover developed by Bledsoe's group at the University of Texas is described. Algorithms are given for its principle routines IMPLY and HOA, and its set of interactive commands are tabulated.

The prover itself (without interaction) is a natural deduction system which uses the concepts of subgoaling, reductions (rewrite rules), procedures, controlled definition instantiation, controlled forward chaining, conditional rewriting and conditional procedures, algebraic simplification, and induction.

It, or variations of it, have been used to prove theorems in set theory and topology, theorems arising from program verification, and limit theorems of calculus and analysis.

Bledsoe, W. W. and Tyson, Mabry, 1975

Typing and Proof by Cases in Program Verification

UT Automatic Theorem Proving Project #15

Ref: A29, A30, A37

PV1, PV7, PV26, PV31

Abstract

Special procedures have been added to an automatic prover to facilitate its handling of inequalities and proof by cases. A data base, called TYPELIST, is used which maintains upper and lower bounds of variables occurring in the proof of a theorem. These procedures have been coded and used to (interactively) prove several theorems arising in automatic program verification.

Sickel, Sharon, 1975

Variable Range Restrictions in Resolution Theorem Proving

University of California, Santa Cruz

Ref: A1, A2, A3, A9, A18, A20, A22, A23, A45

Abstract

The vast number of clauses generated before a proof is found is one of the central problems of classical theorem proving. Many clauses generated by random resolution or resolution restricted by traditional syntactic strategies cannot possibly be used in a proof because the terms that were substituted for the variables in generating the clauses are inconsistent with all possible proofs. Investigating substitutions for variables provides a new way of attacking the problem.

The essence of the approach discussed here is to determine which instances of the original clauses are required for a proof. Information is collected on variables that will specify the ranges of values that they may take on.

We also describe a measure of proof complexity, and <u>all</u> proofs of a given order of complexity are found before the next order is attempted. The solution to a given problem is a set of proof schemas representing all proofs of the lowest complexity possible, rather than a single proof as with traditional methods.

Fishman, D. H., 1975

A Problem-Oriented Search Procedure for Theorem Proving

COINS Tech. Rep. 75C-4, U. Mass.

Ref: A1, A2, A4, A9, A22, A23, A24

Abstract

In this paper, we show that if search procedures are to use problem-specific information to direct a search, then they must have complete control over the deductive process. This includes the freedom to select any pair of clauses to interact as well as the freedom to select the literals upon which to attempt the interaction. We give examples to indicate the nature of inference rules which are incompatible with such search procedures. We then present a framework for a search procedure to indicate where problem-specific information may be utilized. Finally, we augment the search procedure to prevent the generation of certain redundant inferences. The augmentation makes use of lemmas to avoid replicating deduction sequences in solving multiple instances of the same subproblem at distinct places in the search space.

Manna, Zohar and Waldinger, Richard J., 1971

Toward Automatic Program Synthesis

CACM 14,3: 151-165, 1971

Ref: A1, A16/PV15, A47

PV7, PV8, PV18

Abstract

An elementary outline of the theorem-proving approach to automatic program synthesis is given, without dwelling on technical details. The method is illustrated by the automatic construction of both recursive and iterative programs operating on natural numbers, lists, and trees.

In order to construct a program satisfying certain specifications, a theorem induced by those specifications is proved, and the desired program is extracted from the proof. The same technique is applied to transform recursively defined functions into iterative programs, frequently with a major gain in efficiency.

It is emphasized that in order to construct a program with loops or with recursion, the principle of mathematical induction must be applied. The relation between the version of the induction rule used and the form of the program constructed is explored in some detail.

A35

Darlington, J. L., 1968

Automatic Theorem Proving with Equality Substitutions and Mathematical Induction

Machine Intelligence 3: 113-127, 1968

Ref: A1, A9, A13, A23

Review (T. Pavlidis, Princeton, New Jersey)

Usually theorem proving programs use a matching algorithm comparing literals or atomic predicates. This is extended to a method of matching to second order logic where the variables may take predicates as values.

The extended matching is used in obtaining resolvents of formulas and in particular to formulate a principle of mathematical induction. A number of examples are given as well as an implementation of the new matching techniques on an IBM 7090 machine.

A36

Davis, Martin and Putnam, Hilary, 1960

A Computing Procedure for Quantification Theory

JACM 7 : 201-215, 1960

Ref: A17

Local

This classic work represents one of the earliest attempts to develop a uniform proof procedure for quantification theory. The algorithm, when applied to a logically valid formula in the appropriate notation, will terminate and yield a proof of validity; for formulas which are not logically valid, the computation continues indefinitely without giving a result. The formulas are assumed to be quantifier-free and in conjunctive normal form. The method incorporates three rules: a rule for the elimination of one-literal clauses, an "affirmative-negative" rule, and a rule for the elimination of atomic formulas. These rules and the concepts presented have greatly influenced the direction of contemporary theorem-proving techniques.

A37

Bundy, Alan, 1973

Doing Arithmetic with Diagrams

<u>Third IJCAI</u> : 130-138, 1973

Ref: AI, A38

Abstract

A theorem prover for part of arithmetic is described which proves theorems by representing them in the form of a diagram or network. The nodes of this network represent "ideal integers", i.e. objects which have all the properties of integers, without being any particular integer. The links in the network represent relationships between "ideal integers". The procedures which draw these diagrams make elementary deductions based on their built-in knowledge of the functions and predicates of arithmetic. This theorem prover is intended as a model of some kinds of human problem-solving behavior.

A38

Gelernter, H., 1959

Realization of a Geometry-Theorem Proving Machine

<u>Computers and Thought</u> : 134-152

Feigenbaum, E. A. and Feldman, J., eds., McGraw-Hill Book Co., 1963

Ref: A17

Local

This paper represents one of the earliest attempts to implement a mechanical theorem prover. The class of proofs is limited to theorems in elementary euclidean plane geometry. The organization of the geometry machine is divided into three parts: a "syntax computer" and a "diagram computer" both contained in a "heuristic computer". The diagram computer consists of a coordinate representation of the theorem and a series of routines that produce a qualitative description of the diagram. The heuristic computer compares strings generated by the syntactic computer with the diagram interpretation and rejects sequences that are not supported by the diagram. The system generates subgoal trees (with the theorem to be proved at the root) and uses the diagram to indicate which subgoals are probably valid. Sample proofs are included at the end of the discussion.

Stickel, M. E. 1975

A Complete Unification Algorithm for Associative-Commutative Functions

Dept. of Computer Science, Carnegie-Mellon

Ref: A1, A26, A28/PV17

Abstract

An important component of mechanical theorem proving systems are unification algorithms which find most general substitutions which, when applied to two expressions, make them equivalent. Functions which are associative and commutative (such as the arithmetic addition and multiplication functions) are often the subject of mechanical theorem proving. An algorithm which unifies terms whose function is associative and commutative is presented here. The algorithm eliminates the need for axiomatizing the associativity and commutativity properties and returns a complete set of unifiers without recourse to the indefinite generation of variants and instances of the terms being unified required by previous solutions to the problem.

Stickel, Mark E., 1974

The Programmable Strategy Theorem Prover: An Implementation of the Linear MESON Procedure

Dept. of Computer Science, Carnegie-Mellon

Ref: A19, A22

Abstract

The Programmable Strategy Theorem Prover (PSTP) is a theorem proving program for the first order predicate calculus using the linear MESON procedure as the inference system. The linear MESON procedure is a new variant of the model elimination theorem proving procedure for theories with or without equality which is an affirmation rather than a refutation procedure. It is profitably viewed as an extension of the problem-reduction method. The fundamental element of a linear MESON procedure deduction, the chain, is a representation of a set of goals to be solved and their supergoals. PSTP is designed to be used interactively or in a fully automatic mode. Some features of PSTP are a general mechanism for specifying which chains are to be retained and manipulated, an automatic procedure for storing and retrieving information about chains when this information is requested, the capability of specifying an ordering function which can be used for specifying search strategies, and a powerful set of commands. Results of an experiment testing some simple search strategies and comparisons with results from other theorem proving studies are presented.

Morgan, Charles G., 1975

Methods for Automated Theorem Proving in Non-Classical Logics

Dept. of Philosophy, University of Victoria

Ref: A28/PV17

Abstract

In this paper we outline two basic methods for automated theorem proving in non-classical logics, including modal, many-valued, relevance, and intuitionistic logics. We discuss advantages and disadvantages of each method and give several illustrative examples. We outline a procedure for attacking more complex problems using a combination of the two basic methods. Results of experimental applications of the techniques are reported.

Andrews, Peter B., 1975

Refutations by Matings

Dept. of Mathematics, Carnegie-Mellon

Ref: A1, A23, A32

Abstract

Given a refutation by resolution (in first order logic) of a set of occurrences of clauses such that each clause-occurrence is used only once, we define occurrences of literals in the initial clauses to be mated if and only if their descendents are resolved with each other. This leads to an abstract notion of a mating as a relation between occurrences of literals in a given set of clause-occurrences. Conditions are found such that a set S of clauses has no model if and only if some set of occurrences of clauses in S has a mating satisfying these conditions. A mating may be regarded as a plan for a refutation, but once one has an acceptable plan, one need not actually carry out the refutation.

Robinson, J. A., 1968

New directions in mechanical theorem proving

IFIP 68: 63-67

Ref:

Review (R. Anderson and W. W. Bledsoe, Austin, Texas)

This paper presents no new mathematical results on mechanical theorem-proving but instead represents the first step in a campaign by the author to convince the reader that further research in mechanical theorem-proving should be directed toward understanding the omega-order predicate calculus rather than continuing the present investigation of the first-order predicate calculus. The omega-order predicate calculus, also referred to as the simple theory of types, is a formalism which allows one to quantify not only those variables ranging over individuals in one's universe of discourse, but also those ranging over sets of individuals and sets of sets of individuals, etc. This allows one to formulate in a natural manner many mathematical problems which can be formulated in first-order logic only after a detour through set theory, a detour which may be quite difficult (time will tell).

Gödel has proved higher-order logic to be incomplete under the usual meaning of logical validity. However, the author points out that Henkin has in fact shown higher-order logic to be complete if one generalizes the definition of interpretation and thus alters the notion of logical validity. Since the author feels that the Henkin definition of interpretation correctly captures the intuitive notion of interpretation, he argues that this completeness proof should remove any hesitancy the reader may feel about replacing the first-order predicate calculus with the omega-order predicate calculus.

After this presentation of arguments in favor of adopting the omega-order predicate calculus as the basic formalism of mechanical theorem-proving,

a particular formalism in that logic is presented and a direct "semantic tableau" form of proof procedure is outlined. A recent paper by Dag Prawitz [*J. Symbol. Logic* 33, 3 (Sept. 1968)] demonstrates that this proof procedure is complete (in Henkin's sense). However, contrary to a statement in the paper being reviewed, it appears that this direct "cut-free" proof procedure does not yield a true subformula principle since non-subformulas may enter into proofs via λ-expressions.

Since the publication of the paper under review, the author has greatly improved both the formalism and proof procedure for higher-order logic. These improvements will appear in a paper in *Machine Intelligence 4*.

Allen, John, 1973

Preliminary User's Manual For an Interactive Theorem Prover

Stanford Artificial Intelligence Laboratory; Operating Note 73

Ref:

Abstract

This document represents a short guide to using the theorem prover. An earlier version of this program has been described by Allen and Luckham. Many of the later sections of this manual, for pattern matching and subroutining especially, are still in a rudimentary stage of development. Experiments demonstrating various applications of the strategies and the user oriented features are described in a forthcoming report, "Applications of First Order Proof Procedures" by Allen, Luckham, and Morales.

Robinson, J. A., 1968

The generalized resolution principle

Machine Intelligence 3: 77-93

Ref: A1, A9, A18

Review (T. Pavlidis, Princeton, New Jersey)

This paper may well be considered a fundamental one in the area of mathematical theory of automatic theorem proving. It contains further results and generalizes the well known earlier work of the author. The paper contains the following parts:

1) A review of the quantifier-free first-order predicate calculus with equality: Propositions are characterized as "ground" (if among the interpretations satisfying the premise there is none which falsifies the conclusion) and "general" (if only the interpretations whose structural equivalents also satisfy the premise are considered). It is shown that ground propositions are decidable.

2) A study of proofs and inferences for ground propositions: Semantic trees are defined as trees of sentences (for a given proposition P) which satisfy a number of conditions which guarantee that as one goes down a branch of the tree he encounters an increasingly complete description of the vocabulary in which the premises and the conclusions of P are written. Semantic trees are called counter-example trees if they contain failure points.

The first important result of the paper is that from any counter-example tree for a theorem, one obtains automatically a proof of the theorem in which each inference is an application of the generalized ground principle: "from $(A_1 \vee B_1) \cdots (A_k \vee B_k)$ one may infer the 'resolvent' $(A_1 \vee \cdots \vee A_k \vee B)$, whenever B is a disjunction following from $\{B_1, B_2, \cdots, B_k\}$."

3) An extension of the above results to general propositions: This is done

by studying the theory of substitutions, and the generalized resolution principle is stated. It is shown that for any general theorem there is a proof in which each inference is an application of the generalized resolution principle. Finally it is shown that the various earlier forms of the resolution principle are special cases of the present definition.

Robinson, J. A., 1967

Heuristic and Complete Processes in the Mechanization of Theorem Proving

<u>Systems and Computer Science</u>

Univ. of Ontario Press, 1967

Ref: A1, A9, A17, A24, A36, A38

Review (H.D. Block, Ithaca, New York)

Early efforts at theorem-proving algorithms based on the logical completeness of the system suffered from the fact that the number of operations grew so rapidly that the algorithms were impractical for any conceivable computer. On the other hand heuristic methods, based on observation of human problem-solving procedures, do not necessarily arrive at a proof, even if one exists. Moreover, they havenot been very successful at difficult problems.

The present paper presents a clearly written exposition of some methods (the "resolution principle," the "set of support principle," and the "hyper-resolution principle") which are intended to combine the virtues of both the heuristic method and the logically complete algorithms; i.e., the number of operations should be so drastically reduced that the method is practicable, and on the other hand the algorithm will find a proof whenever one exists.

The paper does not mention any computational experimentation, but the method appears to hold great promise.

A47

King, James C. and Floyd, Robert W., 1972

An Interpretation-Oriented Theorem Prover over Integers

Journal of Computer and System Sciences 6: 305-323, 1972

Ref: PV18, PV24

Local

This article describes a special purpose theorem prover which operates exclusively on expressions involving integer variables. The theorem prover is "interpretation oriented" in that it relies heavily on a detailed knowledge of properties of the integers. This knowledge is used throughout the proof process by a formula simplifying system, which manipulates and simplifies the integer expressions, and also by the linear solver, a component which examines interrelations among the expressions. The formula simplifying system, the theorem prover proper, and the linear solver are described in detail, and several sample proofs are included.

A48

Robinson, J. A., 1970

An Overview of Mechanical Theorem Proving

Theoretical Approaches to Non-Numerical Problem Solving: 2-20

Springer-Verlag, 1970

Ref: A1, A2, A3, A4, A9, A10, A11, A13, A17, A18, A23, A24, A26, A35, A36, A38, A43, A45, A46

Local

In this overview, the author sketches the essential concepts of automated theorem proving and indicates research areas where additional work is needed. A generalized and simplified version of Robinson's original resolution principle is presented, and the peculiarities and difficulties of expressing the equality relation are discussed. In addition to an appendix of definitions, an extensive bibliography of pre-1970 reference material is included.

Robinson, J. A., 1970

A Note on Mechanizing Higher Order Logic
<u>Machine Intelligence</u> 5: 123-133, 1970

Ref: A26

Local (Thomas C. Brown)

The author proposes an approach to mechanized reasoning in higher-order logic based on the combinatory logic framework developed by Schönfinkel and later by Curry. He notes the equivalence of the combinatory and the lambda-calculus formalisms and conjectures that the combinatory formalism (based on a binary operation, <u>application</u>, and three constant combinators) would be more appropriate for machine implementations.

The equational basis for combinators is expanded to a forty-seven clause basis (SEM) axiomatizing properties of additional constants such as TRUE, FALSE, NOT, OR AND, IMPLIES, EQUAL, ALL, EXISTS, CHOICE, IF, and FAIL. A conjecture is proved in this system by showing that its negation is inconsistent with SEM, using standard first-order methods based on paramodulation and resolution.

The author proposes to organize long chains of combinatory-logic reasoning into high-level macro-inferences such as normalization (simplification to normal form) and abstraction.

The note concludes with a discussion of possible problems with the approach and directions for subsequent research. In particular, the author wonders whether the <u>type</u> classification of variables should be reimposed on his system, which makes no distinction between propositional and individual variables.

Cooper, D. C., 1971

Programs for Mechanical Program Verification

<u>Machine Intelligence Vol. 6</u> : 43-59

Ref: PV18, PV19

Abstract

A set of programs (written in POP-2) have been developed to be used in investigations into mechanical and mechanically-aided proofs about programs. These include programs for algebraic manipulation and simplification, logic manipulation, input, conversion to flow chart, and conversion to 'block form' of programs, the Presburger decision algorithm, the proof of convergence of programs by Floyd's well-ordering technique, and the proof of correctness of programs by attaching relations to blocks. The present stage of this project will be described.

PV2 - See A8/PV2.

v. Henke, Friedrich W. and Luckham, David C., 1974

Automatic Program Verification III: A Methodology for Verifying Programs

Stanford Artificial Intelligence Laboratory
Memo AIM-256; Report No. STAN-CS-74-474

Ref: A8/PV2, PV40

Abstract

The paper investigates methods for applying an on-line interactive verification system designed to prove properties of PASCAL programs. The methodology is intended to provide techniques for developing a debugged and verified version starting from a program, that (a) is possibly unfinished in some respects, (b) may not satisfy the given specifications, e.g., may contain bugs, (c) may have incomplete documentation, (d) may be written in non-standard ways, e.g., may depend on user-defined data structures.

The methodology involves (i) interactive application of a verification condition generator, an algebraic simplifier and a theorem-prover; (ii) techniques for describing data structures, type constraints, and properties of programs and subprograms (i.e. lower level procedures); (iii) the use of (abstract) data types in structuring programs and proofs.

Within each unit (i.e. segment of a problem), the interactive use is aimed at reducing verification conditions to manageable proportions so that the non-trivial factors may be analysed. Analysis of verification conditions attempts to localize errors in the program logic, to extend assertions inside the program, to spotlight additional assumptions on program subfunctions (beyond those already specified by the programmer), and to generate appropriate lemmas that allow a verification to be completed.

Methods for structuring correctness proofs are discussed that are similar to those of structured programming.

A detailed case study of a pattern matching algorithm illustrating the various aspects of the methodology (including the role of the user) is given.

PV4

Elspas, B., Levitt, K. N., Waldinger, R. J., and Waksman, A., 1972

An Assessment of Techniques for Proving Program Correctness

<u>ACM Computing Surveys</u> 4(2): 97-147, 1972

Ref: A1, A2, A3, A9, A11, A16/PV15, A17, A23, A24, A26, A34, A35, A36, A47
PV7, PV8, PV9, PV10, PV12, PV16, PV18, PV19, PV21, PV22, PV23, PV25

Abstract

The purpose of this paper is to point out the significant quantity of work in progress on techniques that will enable programmers to prove their programs correct. This work has included: investigations in the theory of program schemas or abstract programs; development of the art of the informal or manual proof of correctness; and development of mechanical or semi-mechanical approaches to proving correctness. At present, these mechanical approaches rely upon the availability of powerful theorem-provers, development of which is being actively pursued. All of these technical areas are here surveyed in detail, and recommendations are made concerning the direction of future research toward producing a semi-mechanical program verifier.

PV5

Katz, Shmuel M., and Manna, Zohar, 1973

A Heuristic Approach to Program Verification

<u>Third Int. Joint Conf. on A.I.</u>, 500-512

Ref: PV1, PV18, PV39

Abstract

We present various heuristic techniques for use in proving the correctness of computer programs. The techniques are designed to obtain automatically the "inductive assertions" attached to the loops of the program which previously required human "understanding" of the program's performance. We distinguish between two general approaches: one in which we obtain the inductive assertion by analyzing the predicates which are known to be true at the entrances and exits of the loop (<u>top-down</u> approach), and another in which we generate the inductive assertion directly from the statements of the loop (<u>bottom-up</u> approach).

PV6

Lanzarone, G. A., 1972

Proof of Program Correctness: A Review

Honeywell Computer Journal 6(1): 38-42, 1972

Ref: A16/PV15, A18

PV7, PV9, PV10, PV11, PV12, PV16, PV18, PV19, PV21, PV23

Abstract

The complexity of current software has increased the difficulties of quaranteeing its reliability. Traditional testing methods have proven inadequate to solve the problem. A very promising new approach now being developed consists of attempting to provide a mathematical proof of the correctness of a program.

This article reviews and discusses the various techniques proposed, with their advantages, disadvantages, and practical application.

PV7

Burstall, R. M., 1969

Proving properties of programs by structural induction

Computer Journal 12(1): 41-48, 1969

Ref: PV16, PV18

Abstract

This paper discusses the technique of structural induction for proving theorems about programs. This technique is closely related to recursion induction but makes use of the inductive definition of the data structures handled by the programs. It treats programs with recursion but without assignments or jumps. Some syntactic extensions to Landin's functional programming language ISWIM are suggested which make it easier to program the manipulation of data structures and to develop proofs about such programs. Two sample proofs are given to demonstrate the technique, one for a tree sorting algorithm and one for a simple compiler for expressions.

Manna, Zohar, 1969

The Correctness of Programs

Journal of Computer and System Sciences 3: 119-127, 1969

Ref: PV18

Abstract

This paper is concerned with the relationship between the correctness of programs and the satisfiability (or unsatisfiability) of certain formulas of the first-order predicate calculus. Results on the equivalence of programs are also included.

Dijkstra, E. W., 1968

A Constructive Approach to the Problem of Program Correctness

BIT 8: 174-186, 1968

Ref: PV16, PV18, PV19

Abstract

As an alternative to methods by which the correctness of given programs can be established a posteriori, this paper proposes to control the process of program generation such as to produce a priori correct programs. An example is treated to show the form that such a control might then take. This example comes from the field of parallel programming; the way in which it is treated is representative of the way in which a whole multiprogramming system has actually been constructed.

Hoare, C. A. R., 1969

An Axiomatic Basis for Computer Programming

CACM 12(10): 576-583, 1969

Ref: A16/PV15

PV7, PV18, PV19

Abstract

In this paper an attempt is made to explore the logical foundations of computer programming by use of techniques which were first applied in the study of geometry and have later been extended to other branches of mathematics. This involves the elucidation of sets of axioms and rules of inference which can be used in proofs of the properties of computer programs. Examples are given of such axioms and rules, and a formal proof of a simple theorem is displayed. Finally, it is argued that important advantages, both theoretical and practical, may follow from a pursuance of these topics.

London, Ralph L., 1970

Proving Programs Correct: Some Techniques and Examples

BIT 10: 168-182

Ref: A16/PV15

PV9, PV12, PV16, PV18, PV19

Abstract

Proving the correctness of computer programs is justified as both advantageous and feasible. The discipline of proof provides a systematic search for errors, and a completed proof gives sufficient reasons why the program must be correct. Feasibility is demonstrated by exhibiting proofs of five pieces of code. Each proof uses one or more of the illustrated proof techniques of case analysis, assertions, mathematical induction, standard prose proof, sectioning and a table of variable value changes. Proofs of their programs, some quite lengthy, are cited to support the claim that the techniques work on programs much larger than the examples of the paper. Hopefully, more programmers will be encouraged to prove programs correct.

PV12

London, R. L., 1970

Bibliography on proving the correctness of computer programs

Machine Intelligence 5: 569-580

Ref: A16/PV15
PV7, PV8, PV9, PV10, PV11, PV16, PV18, PV19, PV21, PV22, PV25

Review (from the Preface)

As this bibliography shows, there is considerable research interest in the topic of proving the correctness of computer programs. Included in the bibliography are all published or unpublished papers, books, reports, theses, etc., known to me that are at all related to the topic. Material cited encompasses very abstract papers dealing with the transformation and equivalence of program schemata; proposals and methods for proving individual programs correct; papers proving the correctness of specific algorithms and programs; and work that has proofs as an incidental part of the paper. In other words, the range is from the very abstract to the very practical. An item is included unless it appeared obvious to me that it did not belong. Thus material of marginal relevance is included.

PV13 - See A14/PV13.

PV14 - See A15/PV14.

PV15 - See A16/PV15.

PV16

McCarthy, J. and Painter, J. A., 1967

Correctness of a compiler for arithmetic expressions

Proc. of a Symposium in Applied Mathematics,
Vol. 19: Mathematical Aspects of Computer Science (AMS): 33-41

Ref: A16/PV15

Local

This paper contains a proof of the correctness of a simple compiler for arithmetic expressions involving constants, variables, and binary `+´. The concepts of abstract syntax, state vector, and the use of an interpreter to define the semantics of a programming language are used.

PV17 - See A28/PV17.

Floyd, Robert W., 1967

Assigning Meanings to Programs

Proc. Symp. in Applied Mathematics, Vol. 19: Mathematical Aspects of Computer Science (AMS): 19-32

Ref: A16/PV15

Local

This paper develops a formal methodology for assigning meanings to programs for the purpose of designing proofs of program correctness, equivalence, and termination. The fundamental concept is the association of a proposition with each connective in the flow of control through a program, such that the proposition is true whenever the connective is taken. Proofs of termination involve proving that each step of a program decreases an entity which cannot decrease indefinitely. The notions of correctness and termination are demonstrated on a flowchart language and on a subset of ALGOL.

Naur, Peter, 1966

Proof of algorithms by general snapshots

BIT 6: 310-316, 1966

Ref:

Abstract

A constructive approach to the question of proofs of algorithms is to consider proofs that an object resulting from the execution of an algorithm possesses certain static characteristics. It is shown by an elementary example how this possibility may be used to prove the correctness of an algorithm written in ALGOL 60. The stepping stone of the approach is what is called General Snapshots, i.e. expressions of static conditions existing whenever the execution of the algorithm reaches particular points. General Snapshots are further shown to be useful for constructing algorithms.

PV20

Habermann, A. N., 1975

The Correctness Proof of a Quadratic-Hash Algorithm

Carnegie-Mellon U., Dept. of Computer Science

Ref: PV10, PV18

Abstract

The statements of a program do not always provide sufficient information for proving its correctness. The correctness of the algorithm implemented by the program must often be proved with the pure mathematical techniques or exhaustive enumeration. An example program is presented for which the correctness proof of the program is trivial provided that the correctness of the underlying algorithm can be demonstrated. The program can be viewed as an abstraction of a quadratic hash algorithm. It is used at the end of the paper to encode the algorithm most efficiently.

PV21

de Bakker, J. W., 1969

Semantics of Programming Languages

Advances in Information Systems Science Vol. 2: 173-227

Ref: A16/PV15

PV16, PV18, PV19

Local

This article is a survey of the research on the semantics of programming languages. The first section deals with investigations of basic concepts of programming including program schemata, the axiomatic approach to semantics (due to Igarashi), McCarthy's theory of computation, flow diagrams, and Floyd's method. The second section concerns the formal definition of programming languages and briefly surveys the Markov-Algorithm approach, McCarthy's Formalism, the Vienna method, and the λ - Calculus approach. Little background material is provided, although an extensive bibliography is included.

PV22

Florentin, J. J., 1968

Language Definition and Compiler Validation

Machine Intelligence 3: 33-41, 1968

Ref: PV16

Abstract

A method is proposed for checking that a compiler translates in accordance with the user language definition. It involves formalizing the compiler action into a series of automata models, each with a state transition table. A finite number of input-output experiments are then to be done to recover the state transition tables, and these are compared to the specifications. In a language with operating system facilities, details of the operating system must be included in the automata states. The problems arising here are briefly discussed.

PV23

Elspas, B., Green, M. W., and Levitt, Karl, 1971

Software Reliability

Computer 4, 1: 21-27, 1971

Ref: A1, A34, A47

PV8, PV9, PV10, PV18, PV19

Local

This paper examines a number of approaches to improving the writing, debugging, and verification of programs. The three major topics considered are improvements in the design of programming languages, the semantic checking of programs at compile time, and automatic program verification.

PV24

King, J. C., 1972

A Program Verifier

IFIP 71: 234-249, 1972

Ref: A47

PV12, PV18, PV25

Abstract

This paper describes a computer program, called a program verifier, which proves the correctness of computer programs. In order to show that a program is correct, it must be annotated with propositions in a symbolic notation which define the correct relations among the program variables. The verifier then checks for consistency between the program and its propositions. Several interesting programs have been automatically verified by this verifier including a simple sort program, a program which checks an integer for "prime", and a subtle program for raising an integer to an integral power.

PV25

Good, Donald I., 1970

Toward a Man-Machine System for Proving Program Correctness

Ph.D. Thesis, Computer Sciences Dept., Univ. of Wisconsin, 1970

Ref: A16/PV15

PV7, PV8, PV10, PV12, PV18, PV19, PV24

Abstract

The problem considered is the realization of a computerized program proving system that is applicable to real computer programs. The input to such a system is both a program and the specifications this program must meet in order to be correct. If the program actually is correct, then the system should either prove that the program is correct, or at least provide enough assistance so that proving correctness becomes a feasible task for a human. If the program is not correct, the system should assist in locating errors in the program.

The total question of the practical realization of proving the correctness of programs is considered from three levels, of decreasing generality. At the first level, a theoretical foundation is developed. A simple computational model that can describe a significant class of real programs is defined. Based on this model, the idea of a result of a program is formalized, several general types of results are defined and several properties of results are proved. These properties provide a useful, formal basis for further consideration of the correctness problem. This theoretical foundation also unifies some of the results obtained previously by others.

The second level of consideration is a general method for proving program results, the inductive assertion method. It is shown how the "validity" of this method can be proved from the theoretical foundation

developed previously. Also, the general applicability of this method is proved. A crucial step in the inductive assertion method is the construction of verification conditions. Several forms of verification conditions are derived and particular attention is given to their automatic construction from the program.

The third level of consideration is the realization of a man-machine, interactive program proving system through a partial automation of the inductive assertion method. This discussion is based on the actual programming of key parts of such a system. In this system the machine provides services to the human who is ultimately responsible for constructing the correctness proof. The use of this type of system on real programs presents some unique problems in the application of the inductive assertion method.

Waldinger, R. J., and Levitt, K. N., 1974

Reasoning About Programs

Artifical Intelligence 5: 235-316, 1974

Ref: A1, A8/PV2

PV4, PV5, PV10, PV16, PV18, PV19, PV39, PV40

Abstract

This paper describes a theorem prover that embodies knowledge about programming constructs, such as numbers, arrays, lists, and expressions. The program can reason about these concepts and is used as part of a program verification system that uses the Floyd-Naur explication of program semantics. It is implemented in the QA4 language; the QA4 system allows many pieces of strategic knowledge, each expressed as a small program, to be coordinated so that a program stands forward when it is relevant to the problem at hand. The language allows clear, concise representation of this sort of knowledge. The QA4 system also has special facilities for dealing with commutative functions, ordering relations, and equivalence relations; these features are heavily used in this deductive system. The program interrogates the user and asks his advice in the course of a proof. Verifications have been found for Hoare's FIND program, a real-number division algorithm, and some sort programs, as well as for many simpler algorithms. Additional theorems have been proved about a pattern matcher and a version of Robinson's unification algorithm. The appendix contains a complete, annotated listing of the deductive system and annotated traces of several of the deductions performed by the system.

PV27

Liskov, B. H., and Zilles, S., 1975

Specification Techniques for Data Abstractions

IEEE Trans. Software Engineering Vol. SE-1; 1: 7-19, 1975

Ref: PV10, PV18, PV36

Abstract

The main purposes in writing this paper are to discuss the importance of formal specifications and to survey a number of promising specification techniques. The role of formal specifications both in proofs of program correctness, and in programming methodologies leading to programs which are correct by construction, is explained. Some criteria are established for evaluating the practical potential of specification techniques. The importance of providing specifications at the right level of abstraction is discussed, and a particularly interesting class of specification techniques, those used to construct specifications of data abstractions, is identified. A number of specification techniques for describing data abstractions are surveyed and evaluated with respect to the criteria. Finally, directions for further research are indicated.

PV28

McKeeman, W. M., 1975

On Preventing Programming Languages from Interfering with Programming

IEEE Trans. Software Engineering Vol. SE-1,1: 19-26, 1975

Ref: PV10

Abstract

Wirth has proposed a method of "stepwise refinement" for writing computer programs. This paper proposes that the steps be expressed as proofs. A program for the eight-queens problem is developed, and the proof method is applied across two of the steps of the development. The strengths and weaknesses of the method, and its implications for the programming process and programming language design are discussed.

Hewitt, Carl E. and Smith, Brian, 1975

Towards a Programming Apprentice

IEEE Trans. Software Engineering Vol. SE-1: 26-45, 1975

Ref: A8/PV2

PV5, PV7, PV18, PV19, PV26, PV39, PV40, PV42

Abstract

The PLANNER Project is constructing a "programming apprentice" to assist in knowledge based programming. We would like to provide an environment which has substantial knowledge of the semantic domain for which the programs are being written, and knowledge of the purposes that the programs are supposed to satisfy. Further, we would like to make it easy for the programmer to communicate the knowledge about the program to the apprentice. The apprentice is to aid expert programmers in the following kinds of activities:

(1) establishing and maintaining consistency of specifications;
(2) validating that modules meet their specifications;
(3) answering questions about dependencies between modules;
(4) analyzing implications of perturbations in modules; and
(5) analyzing implications of perturbations in specifications.

We use "contracts" (procedural specifications) to express what is supposed to be accomplished as opposed to how it is supposed to be done. The idea is that at least two procedures should be written for each module in a system. One procedure implements a method for accomplishing a desired transformation and the other can check that the transformation has in fact been accomplished. The programming apprentice is designed for interactive use by expert programmers in the meta-evaluation of implementations in the context of their contracts and background knowledge. Meta-evaluation

produces a "justification" which makes explicit exactly how the module depends on the contracts of other modules and on the background knowledge. The justification is used in answering questions on the behavioral dependencies between modules and in analyzing the implications of perturbations in specifications and/or implementation.

Ramamoorthy, C. V. and Ho, S. B. F., 1975

Testing Large Software with Automated Software Evaluation Systems

<u>IEEE Trans. Software Engineering</u> Vol. SE-1: 46-58, 1975

Ref: PV4

Abstract

In the past few years, research has been actively carried out in an attempt to improve the quality and reliability of large-scale software systems. Although progress has been made on the formal proof of program correctness, proving large-scale software systems correct by formal proof is still many years away. Automated software tools have been found to be valuable in improving software reliability and attacking the high cost of software systems. This paper attempts to describe some main features of automated software tools and some software evaluation systems that are currently available.

Good, Donald I., London, Ralph L., and Bledsoe, W. W., 1975

An Interactive Program Verification System

<u>IEEE Trans. Software Engineering</u> Vol. SE-1, 1: 59-67, 1975

Ref: A8/PV2, A12, A29, A31, A37

PV3, PV26, PV40

Abstract

This paper is an initial progress report on the development of an interactive system for verifying that computer programs meet given formal specifications. The system is based on the conventional inductive assertion method: given a program and its specifications, the object is to generate the verification conditions, simplify them, and prove what remains. The important feature of the system is that the human user has the opportunity and obligation to help actively in the simplifying and proving. The user, for example, is the primary source of problem domain facts and properties needed in the proofs. A general description is given of the overall design philosophy, structure, and functional components of the system, and a simple sorting program is used to illustrate both the behavior of major system components and the type of user interaction the system provides.

PV32

German, Steven M. and Wegbreit, Ben, 1975

A Synthesizer of Inductive Assertions

<u>IEEE Trans. Software Engineering</u> Vol. SE-1, 1: 68-75, 1975

Ref: A8/PV2

PV1, PV4, PV5, PV18, PV26, PV40

Abstract

Most current methods for mechanical program verification require a complete inductive assertion on each loop. As this is tedious and error prone, producing a program with complete, correct assertions is moderately difficult. This paper describes a prototype system VISTA which provides assistance in synthesizing correct inductive assertions. Given only the source program, it is able to generate a useful class of assertions automatically. For a larger class, it is able to extend partial inductive assertions supplied by the programmer to form complete assertions from which it proves program correctness. Its synthesis methods include: symbolic evaluation in a weak interpretation, combining output assertions with loop assertions, and extracting information from proofs which fail in order to determine how assertions should be strengthened. We present VISTA as a step toward practical program verifiers.

PV33

Basu, Sanat K. and Misra, Jayadev, 1975

Proving Loop Programs

<u>IEEE Trans. Software Engineering</u> Vol. SE-1, 1: 76-86, 1975

Ref: A8/PV2

PV4, PV5, PV10, PV18, PV31, PV39

Abstract

Given a "DO WHILE" program P and a function F on a domain D, we investigate the problem of proving (or disproving) if P computes F over D. We show that if P satisfies certain natural constraints (well behaved), then there is a loop assertion, independent of the structure of the loop body, that is both necessary and sufficient for proving the hypothesis. We extend these results to classes of loop programs which are not well behaved and to FOR loops. We show the sufficiency of Hoare's DO WHILE axiom for well-behaved loop programs. Applications of these ideas to the problem of mechanical generation of assertions is discussed.

PV34

Owicki, Susan and Gries, David, 1975

Proving Properties of Parallel Programs: An Axiomatic Approach

Dept. of Computer Science, Cornell University, TR75-243

Ref: PV10

Abstract

This paper presents an axiomatic technique for proving a number of properties of parallel programs. Hoare has given a set of axioms for partial correctness of parallel programs, but they are not strong enough in most cases. Here we define a deductive system which is in some sense complete for partial correctness. The information in a partial correctness proof is then used to prove such properties as mutual exclusion, blocking, and termination.

PV35

Lauer, P., 1971

Consistent Formal Theories of the Semantics of Programming Languages

IBM Laboratory Vienna TR 25.121, 1971

Ref: A16/PV15, A34

PV7, PV8, PV10, PV18, PV21

Abstract

An example high-level programming language is introduced by means of a definition using a Backus Naur Form production system and informal but technical English. The semantics of the language is rendered exact by means of two interpretive models consisting of memory states, control states and interpretive functions for the language. The two models are shown to be equivalent. The semantics of the same language is characterized by means of an axiomatic theory of relations of memory states. The interpretive models are shown to be models of the axiomatic theory. The notion of conditional correctness of programs of the example language is explicated by means of a deductive theory of relations of propositional expressions. The deductive theory is shown to be consistent relative to the axiomatic theory proving that the interpretive models are models of the deductive theory also. A brief account of the history and results of the study is given, relating it to some of the alternative theories of formal semantics of programming languages in the field.

PV36

Birman, A., 1974

On Proving Correctness of Microprograms

IBM J. Research & Dev. 18,3: 250-266, 1974

Ref: PV18

Abstract

This paper describes the results of an investigation in proving the correctness of microprograms. The vehicle used is the S-machine, which is a very simple "paper" computer. The approach to the proof of correctness is based on formally defining the machine-instruction level and the microprogramming level of the given machine, and then showing that these "interfaces" are equivalent through the use of a concept called algebraic simulation.

PV37

Leeman, George B., Carter, William C., and Birman, Alexander, 1974

Some Techniques for Microprogram Validation

IFIP 74: 76-80, 1974

Ref: A16/PV15

PV18, PV36

Abstract

This paper presents procedures which can be used to prove the correctness of microprograms. Our techniques are applied in conjunction with a method described in Birman (PV36). We discuss the concept of algebraic simulation between abstract programs, and we demonstrate how simulation proofs can be carried out, simplified, and made more efficient. We also illustrate how the well-known method of inductive assertions can be incorporated into our approach. Our techniques are illustrated for a simple machine, and they have been used for more complex machines as well.

PV38

Leeman, George B., Jr., 1975

Some Problems in Certifying Microprograms

IEEE Trans. Comp. Vol. C-24,5: 545-553, 1975

Ref: PV18, PV36, PV37

Abstract

A hypothetical computer is described, and procedures are indicated for showing the correctness of its microprogram. The underlying method used is that of Birman (PV36). However, the computer discussed has some realistic characteristics not shared by the machine treated in (PV36), and the details of the microcode validation are complicated by this fact. A formal technique for partitioning the proof is presented and illustrated with examples.

PV39

Wegbreit, Ben, 1973

Heuristic Methods for Mechanically Deriving Inductive Assertions

Third IJCAI: 524-536, 1973

Ref: PV1, PV4, PV18

Abstract

Current methods for mechanical program verification require a complete predicate specification on each loop. Because this is tedious and error-prone, producing a program with complete, correct predicates is reasonably difficult and would be facilitated by machine assistance. This paper discusses heuristic methods for mechanically deriving loop predicates from their boundary conditions and for mechanically completing partially specified loop predicates.

PV40

Igarashi, S., London, R. L., and Luckham, D. C., 1973

Automatic Program Verification I: A Logical Basis and its Implementation

Stanford Artificial Intelligence Laboratory Report No. STAN-CS-73-365

Ref: PV10, PV16, PV18, PV24, PV42

Abstract

Defining the semantics of programming languages by axioms and rules of inference yields a deduction system within which proofs may be given that programs satisfy specifications. The deduction system herein is shown to be consistent and also deduction complete with respect to Hoare's system. A subgoaler for the deduction system is described whose input is a significant subset of Pascal programs plus inductive assertions. The output is a set of verification conditions or lemmas to be proved. Several non-trivial arithmetic and sorting programs have been shown to satisfy specifications by using an interactive theorem prover to automatically generate proofs of the verification conditions. Additional components for a more powerful verification system are under construction.

PV41

Aiello, L., Aiello, M., and Weyhrauch, R. W., 1974

The Semantics of PASCAL in LCF

Stanford Artificial Intelligence Laboratory
Report No. STAN-CS-74-447

Ref: PV8, PV10, PV16, PV18, PV26, PV40

Abstract

We define a semantics for the arithmetic part of PASCAL by giving it an interpretation in LCF, a language based on the typed λ-calculus. Programs are represented in terms of their abstract syntax. We show sample proofs, using LCF, of some general properties of PASCAL and the correctness of some particular programs. A program implementing the McCarthy Airline reservation system is proved correct.

PV42

Floyd, Robert W., 1971

Toward Interactive Design of Correct Programs

IFIP 71: 7-10

Ref: A1, A34, A47
PV8, PV18

Review (R. L. London, Marina del Rey, California)

This invited paper is a short, well-written, and convincing presentation to show that the computer could substantially assist in the design of correct programs. Much of the paper is an imagined interaction between programmer and machine working jointly to construct a program to locate a symbol in an ordered table. It is shown how the "computer is continually checking the program, at each level of specification, for consistency with the programmer's stated intentions." The proof method of inductive assertions is applied to aid in the *design* of the program. In the interaction, the machine deduces a counterexample to a program; an appendix shows how this was done.

The author properly states, "Rather than randomly generate consequences of what is known, the system must have a strong sense of direction and must aim for simplification." He also suggests several areas in which to develop needed skills of mathematical manipulation by computer.

The reviewer is aware of a recent piece of work, as yet unpublished, which further indicates the author's imagined interaction to be very realistic. It seems only fair, however, to let that person announce the results.

PV43

Gerhart, Susan L., 1975

On the Relationship Between Structured Programming and Correctness Proofs

ICASE Report 75-15, NASA Langley Research Center

Ref: PV27, PV41

Abstract

This paper develops the view that there are two distinct theories, a mathematical theory of programs and an explanatory theory of programming, which are evolving from and which subsume the respective subjects of correctness proofs of programs and structured programming. The relationship between the two subjects is explored in terms of this view. Three specific topics are used to illustrate the similarities and differences in the theories: the goto statement of programming languages, the nature and use of abstraction and the fundamental problem of program specification.

Kang, Andy N. C. and Wang, Shih-Ho, 1975

An Extension of the Direct Method for Verifying Programs

Dept. of Computer Science, Virginia Polytechnic Institute
and State University, TR CS75022-R

Ref: A28/PV17
PV5

Abstract

A direct method based on a finite set of path formulas which describe the input-output relations of a given program can be used to verify programs containing no overlapping loops. One major difficulty in verifying programs with overlapping loops using the above method is that too many path formulas (possibly infinite) needed to be considered. In this paper, we circumvent the above difficulty by applying the concept of modularity. The idea is to divide a program with overlapping loops into several small modules so that each module contains no overlapping loop. This can always be achieved if the program is in structured form. Then the path formulas will be derived for each module. By combining the path formulas for the modules, one can further obtain the path formulas for the given program and then use them to verify the program.