(4)

DEDUCTIVE PROGRAMMING

Final Technical Report:
Department of the Navy
Contract *N00039-84-C-0211* (task 3)
Expiration Date: May 31, 1987

by
Zohar Manna, Professor
Computer Science Department
Stanford University
Stanford, California 94305

Sponsored by

Defense Advanced Research Projects Agency (DoD)
1400 Wilson Boulevard
Arlington, Virginia 22209-2389

"Deductive Programming"

Issued by Space and Naval Warfare Systems Command

Under Contract #N00039-84-C-0211, Task 3

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle) Deductive Programming	5. TYPE OF REPORT & PERIOD COVERED final technical report 6/18/84-6/17/87
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Zohar Manna	8. CONTRACT OR GRANT NUMBER(s) N00039-84-C-0211, Task 3
9. PERFORMING ORGANIZATION NAME AND ADDRESS Computer Science Department Stanford University Stanford, CA 94305	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS SPAWAR 3241C2 Space and Naval Warfare Systems Command Washington, D.C. 20363-5100	12. REPORT DATE November 1988
	13. NUMBER OF PAGES 7
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) ONR Representative - Mr. Robin Simpson 202 McCullough Stanford University Stanford, CA 94305	15. SECURITY CLASS. (of this report) Unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release: distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

See attached report.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

S/N 0102-LF-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

TECHNICAL SUMMARY

Our research concentrated on the following topics:

- **Special Relations in Automated Deduction** ([MW1])

Theorem provers have exhibited super-human abilities in limited, obscure subject domains but seem least competent in areas in which human intuition is best developed. One reason for this is that an axiomatic formalization requires us to state explicitly facts that a person dealing in a familiar subject would consider too obvious to mention; the proof must take each of these facts into account explicitly. A person who is easily able to construct an argument informally may be too swamped in detail to understand, let alone produce, the corresponding formal proof. A continuing effort in our research is to make formal theorem proving more closely resemble intuitive reasoning. One case in point is our treatment of special relations.

In most proofs of interest for program synthesis, certain mathematical relations, such as equality and orderings, present special difficulties. These relations occur frequently in specifications and in derivation of proofs. If their properties are represented axiomatically, proofs become lengthy, difficult to understand, and even more difficult to produce or discover automatically. Axioms such as transitivity have many consequences, most of which are irrelevant to the proof; including them produces an explosion in the search space.

For the equality relation, the approach that was adopted early on is to represent its properties with rules of inference rather than axioms. In resolution systems, two rules of inference, paramodulation (Wos and Robinson) and E-resolution (Morris), were introduced. Proofs using these rules are shorter and clearer, because one application of a rule can replace the application of several axioms. More importantly, we may drop the equality axioms from the clause set, thus eliminating their numerous consequences from the search space.

We have discovered two rules of inference that play a role for an arbitrary relation analogous to that played by paramodulation and E-resolution for the equality relation. These rules apply to sentences employing a full set of logical connectives; they need not be in the clause form required by traditional resolution theorem provers. We intend both these rules to be incorporated into theorem provers for program synthesis.

Employing the new special-relations rules yields the same benefits for an *arbitrary* relation as using paramodulation and E-resolution yields for equality: proofs become shorter and more comprehensible and the search space becomes sparser.

- **Binary-Search Algorithms** ([MW2])

Some of the most efficient numerical algorithms rely on a *binary-search* strategy; according to this strategy, the interval in which the desired output is sought is divided roughly in half at each iteration. This technique is so useful that some authors (e.g., Dershowitz and Manna, and Smith) have proposed that a general binary-search paradigm or schema be built into program synthesis systems and then specialized as required for particular applications.

It is certainly valuable to store such schemata if they are of general application and difficult to discover. This approach, however, leaves open the question of how schemata are discovered in the first place. We have found that the concept of binary search appears quite naturally and easily in the derivations of some numerical programs. The concept arises as the result of a single resolution step, between a goal and itself, using our deductive-synthesis techniques (Manna and Waldinger [80]).

The programs we have produced in this way (e.g., real-number quotient and square root, integer quotient and square root, and array searching) are quite simple and reasonably efficient, but are bizarre in appearance and different from what we would have constructed by informal means. For example, we have developed by our synthesis techniques the following real-number square-root program $sqrt(r, \epsilon)$:

$$sqrt(r,\, \epsilon) \;\Leftarrow\; \begin{cases} \mathit{if}\;\; max(r,\, 1) < \epsilon \\ \mathit{then}\;\; 0 \\ \mathit{else}\;\; \mathit{if}\;\; \left[sqrt(r,\, 2\epsilon) + \epsilon\right]^2 \le r \\ \qquad \mathit{then}\;\; sqrt(r,\, 2\epsilon) + \epsilon \\ \qquad \mathit{else}\;\; sqrt(r,\, 2\epsilon). \end{cases}$$

The program tests if the error tolerance ϵ is sufficiently large; if so, 0 is a close enough approximation. Otherwise, the program finds recursively an approximation within 2ϵ less than the exact square root of r. It then tries to refine this estimate, increasing it by ϵ if the exact square root is large enough and leaving it the same otherwise.

This program was surprising to us in that it doubles a number rather than halving it as the classical binary-search program does. Nevertheless, if the repeated occurrences of the recursive call $sqrt(r, 2\epsilon)$ are combined by common-subexpression elimination, this program is as efficient as the familiar one and somewhat simpler.

- **A Theory of Plans** ([MW3][MW4])

Problems in commonsense and robot planning were approached by methods adapted from our program-synthesis research; planning is regarded as an application of automated deduction. To support this approach, we introduced a variant of situational logic (Manna and Waldinger [81]), called *plan theory*, in which plans are explicit objects. A machine-oriented deductive-tableau inference system is adapted to plan theory. Equations and equivalences of the theory are built into a unification algorithm for the system. Frame axioms are built into the resolution rule.

Special attention was paid to the derivation of conditional and recursive plans. Inductive proofs of theorems for even the simplest planning problems, such as clearing a block, have been found to require challenging generalizations.

- **Deductive Synthesis of Dataflow Networks** ([JMW])

The synthesis of concurrent programs is much more complicated than the synthesis of sequential programs. In general, a concurrent program does not have a single input value and a single output value, but receives several inputs and sends several outputs during its execution. If we consider *sequences* of input and output values, then we can specify a concurrent program by giving a relation between the sequence of input values and the sequence of output values. This specification method is natural especially for networks of deterministic processes that communicate asynchronously by sending messages over buffered channels. Deterministic data flow networks fall into this category.

We have developed a method for the deductive synthesis of deterministic dataflow networks, which are specified by a relation between sequences of input values and sequences of output values.

Our synthesis method consists of two stages. The first stage, the deductive-synthesis stage, starts from a specification of the network. Using the deductive-tableau techniques of Manna and Waldinger [80], a system of recursive equations is synthesized. This system can be regarded as an applicative program that satisfies the specification for the network, but it does not directly represent any structure or parallelism of a network. In the second stage, the system of recursive equations is transformed into a dataflow network.

• Logic: The Calculus of Computer Science ([MW5])

The research papers in which we have presented the deductive approach to program synthesis has been addressed to the usual academic readers of the scholarly journals. In an effort to make this work accessible to a wider audience, including computer science undergraduates and programmers, we have developed a more elementary treatment in the form of a two-volume book, *The Logical Basis for Computer Programming*, Addison-Wesley (Manna and Waldinger [85c]).

This book requires no computer programming and no mathematics other than an intuitive understanding of sets, relations, functions, and numbers; the level of exposition is elementary. Nevertheless, the text presents some novel research results, including

- theories of strings, trees, lists, finite sets and bags, which are particularly well suited to theorem-proving and program-synthesis applications;
- formalizations of parsing, infinite sequences, expressions, substitutions, and unification;
- a nonclausal version of skolemization;
- a treatment of mathematical induction in the deductive-tableau framework.

• Verification of Concurrent Programs ([MP1])

We studied in detail the proof methodologies for verifying temporal properties of concurrent programs. Corresponding to the main classification of temporal properties into the classes of *safety* and *liveness* properties, appropriate proof principles were presented for each of the classes.

We developed proof principles for the establishment of *safety* properties. We showed that essentially there is only one such principle for safety proofs, the invariance principle, which is a generalization of the method of intermediate assertions. We also indicated special cases under which these assertions can be found algorithmically.

The proof principle that we developed for *liveness* properties is based on the notion of well-founded descent of ranking functions. However, because of the nondeterminancy inherent in concurrent computations, the well-founded principle must be modified in a way that is strongly dependent on the notion of *fairness* that is assumed in the computation. Consequently, three versions of the well-founded principle were presented, each corresponding to a different definition of fairness.

• A Resolution Approach to Temporal Proofs ([A][AM1][AM2])

A novel proof system for temporal logic was developed. The system is based on the classical non-clausal resolution method, and involves a special treatment of quantifiers and temporal operators.

Soundness and completeness issues of resolution and other related systems were investigated. While no effective proof method for temporal logic can be complete, we established that a simple extension of the resolution system is as powerful as Peano Arithmetic.

The use of temporal logic as a programming language was explored. We suggested that a specialized temporal resolution system could effectively interpret programs written in a restricted version of temporal logic.

We also provided analogous resolution systems for other useful modal logics, such as certain modal logics of knowledge and belief.

Specification and Verification by Predicate Automata ([MP2])

We examined the possibility of specifying and verifying temporal properties using an extension of finite-state automata, called ***predicate automata***. These automata extend the conventional notion of automata in three respects. The first extension is that the conditions for transitions between states can be arbitrary predicates expressed in a first-order language. The second extension is that these automata inspect *infinite* input sequences, and hence a more complex acceptance criterion is needed. The third extension is that non-determinism is interpreted *universally*, rather than *existentially*, as is the case in conventional non-deterministic finite-state automata. This means that if the automata can generate several possible runs, in response to a given input, then it is required that *all* runs are accepting.

By introducing conventions for representing automata in a structured form, we demonstrated that specification of temporal properties by automata can become very legible and understandable, and presents a viable alternative to their formulation in temporal logic.

A single proof rule was presented for proving that a given program satisfies a property specifiable by a predicate automaton. The rule was shown to be sound and relatively complete.

A Hierarchy of Temporal Properties ([MP3])

We proposed a classification of temporal properties into a hierarchy which refines the known *safety-liveness* classification of properties. The classification is based on the different ways a property of finite computations can be extended into a property of infinite computations.

This hierarchy was studied from three different perspectives, which were shown to agree. Respectively, we examined the cases in which the finitary properties, and the infinitary properties extending them, are unrestricted, specifable by temporal logic, and specifiable by predicate automata. The unrestricted view leads also to a topological characterization of the hierarchy as occupying the lowest two levels in the Borel hierarchy.

For properties that are expressible by temporal logic and predicate automata, we provide a syntactic characterization of the formulae and automata that specify properties of the different classes. The temporal logic characterization strongly relies on the use of the past temporal operators.

Corresponding to each class of properties, we presented a proof principle that is adequate for proving the validity of properties in that class.

Logic Programming Semantics: Techniques and Applications ([B1]-[B3])

It is generally agreed that providing a precise formal semantics for a programming language is helpful in fully understanding the language. This is especially true in the case of logic-programming-like languages for which the underlying logic provides a well-defined but insufficient semantic basis. Indeed, in addition to the usual model-theoretic semantics of the logic, proof-theoretic deduction plays a crucial role in understanding logic programs. Moreover, for specific implementations of logic programming, e.g. PROLOG, the notion of deduction stategy is also important.

We provided semantics for two types of logic programming languages and develop applications of these semantics. First, we propose a semantics of PROLOG programs that we use as the basis of a proof method for termination properties of PROLOG programs. Second, we turn to the temporal logic programming language TEMPLOG of Abadi and Manna, develop its declarative semantics, and then use this semantics to prove a completeness result for a fragment of temporal logic and to study TEMPLOG's expressiveness.

In our PROLOG semantics, a program is viewed as a function mapping a goal to a finite or infinite sequence of answer substitutions. The meaning of a program is then given by the least solution of a system of functional equations associated with the program. These equations are taken as axioms in a first-order theory in which various program properties, especially termination or non-termination properties, can be proved. The method extends to PROLOG programs with extra-logical features such as *cut*.

For TEMPLOG, we provide two equivalent formulations of the declarative semantics: in terms of a minimal temporal Herbrand model and in terms of a least fixpoint. Using the least fixpoint semantics, we are able to prove that TEMPLOG is a fragment of temporal logic that admits a complete proof system. This semantics also enables us to study TEMPLOG's expressiveness. For this, we focus on the propositional fragment of TEMPLOG and prove that the expressiveness of propositional TEMPLOG queries essentially corresponds to that of finite automata.

REFERENCES

Research papers and Ph.D. theses supported by this contract.

* indicates papers that are attached as part of this report.

[A] M. Abadi (supervised by Z. Manna), *Temporal-logic theorem proving*, Ph.D. Thesis, Computer Science Dept., Stanford University (1986).

*[AM1] A. Abadi and Z. Manna, "Modal theorem proving," *8th International Conference on Automated Deduction*, Oxford, England, Lecture Notes in Computer Science 230 (R. Parikh, ed.), Springer-Verlag, July 1986, pp. 172–189.

[AM2] A. Abadi and Z. Manna, "Nonclausal deduction in first-order temporal logic," submitted to the *JACM* 1988.

[B1] M. Baudinet, "Proving Termination Properties of PROLOG Programs: A Semantic Approach", *Proceedings of the Third Annual Symposium on Logic in Computer Science*,, pp. 336–347, Edinburgh, Scotland, July 1988.

*[B2] M. Baudinet, "Temporal Logic Programming is Complete and Expressive", *Proceedings of the Sixteenth ACM Symposium on Principles of Programming Languages*, Austin, Texas, January 1989.

[B3] M. Baudinet (supervised by Z. Manna), *Logic Programming Semantics: Techniques and Applications*, Ph.D. Thesis, Computer Science Dept., Stanford University (1989).

*[JMW] B. Jonsson, Z. Manna and R. Waldinger, "Towards deductive synthesis of data-flow networks," *Symposium on Logic in Computer Science*, Cambridge, MA, June 1986, pp. 26–37.

[MP1] Z. Manna and A. Pnueli, *Temporal Specification and Verification of Concurrent Program*, Textbook, to appear 1989.

*[MP2] Z. Manna and A. Pnueli, "Specification and verification of concurrent programs by ∀-automata," *14th Symposium on Principles of Programming Languages*, Munich, Jan. 1987, pp. 1–12.

[MP3] Z. Manna and A. Pnueli, "A hierarchy of temporal properties," To appear in the *Proceedings of Colloquium on Temporal Logic and Specification* (B. Banieqbal and H. Barringer), Lecture Notes in Computer Science, Springer Verlag, 1989.

*[MW1] Z. Manna and R. Waldinger, "Special relations in automated deduction," *Journal of the ACM*, Vol. 33, No. 1 (Jan. 1986), pp. 1–59.

*[MW2] Z. Manna and R. Waldinger, "The origin of the binary-search paradigm," *Science of Computer Programming* Journal, Vol. 9, No. 1 (August 1987), pp. 37–83.

*[MW3] Z. Manna and R. Waldinger, "How to clear a block: A theory of plans," in *Reasoning About Actions and Plans: Proceedings of the 1986 Workshop*, Timberline, Oregon, July 1986, Morgan and Kaufmann (M.P. Georgeff and A.L. Lansky, eds.), pp. 11–45. Also in the *Journal of Automated Reasoning*, Vol. 3, No. 4 (December 1987), pp. 343–377.

[MW4] Z. Manna and R. Waldinger, "The Deductive Synthesis of Imperative LISP Programs," AAAI Conference, Seattle, July 1987.

[MW5] Z. Manna and R. Waldinger, *Logical Basis for Computer Programming*, Textbook, Addison-Wesley Pub., Volume 2: Deductive Systems (to appear, 1989).