②

SRI International

# PLANNING AND IMPERATIVE PROGRAM SYNTHESIS: A DEDUCTIVE APPROACH

ONR Project N00014-89-K-0114
SRI Project 7433
Final Technical Report

March 26, 1990

By: **Richard Waldinger, Principal Scientist**
Artificial Intelligence Center
Computing and Engineering Sciences Division

Prepared for:
Office of Naval Research
800 North Quincy Street
Arlington, Virginia 22217
**Attention:** Dr. Alan Meyrowitz

APPROVED:

[illegible]

C. Raymond Perrault, Director
Artificial Intelligence Center

Donald L. Nielson, Vice President and Director
Computing and Engineering Sciences Division

SRI International

## Contents

## 1 Summary of Work Accomplished

Under the support of this contract, SRI International (SRI) has extended a deductive approach to the synthesis of programs to the derivation of imperative programs and plans. An interactive implementation of the technique has been developed. The approach has also been applied to the derivation of programs for database updating. A book describing the deductive approach has been completed.

### 1.1 Background

For several years we have been working (largely under ONR and NSF support) on the automatic *synthesis* of computer programs. This is the task of deriving a program to meet the conditions of a given specification. We have settled on a deductive approach [MW80] to this problem, according to which programming is regarded as a task in deduction, or theorem proving. To construct a program, we prove the existence of a data object (e.g., a number or list) meeting the specified conditions. The proof is restricted to be sufficiently constructive to indicate a computational method (or algorithm) for finding the desired object. This method becomes the basis for the program that we then extract from the proof.

The theorem is proved in a *background theory*, which provides the properties of the data objects as well as the constructs available in the target programming language. Programs constructed in this way are guaranteed to meet their specifications; the derivation constitutes a formal verification of the program. The structure of the proof is reflected in the structure of the corresponding extracted program. In particular, case analysis in the proof produces a conditional test in the program; mathematical induction in the proof produces a recursive call in the program; and the use of subsidiary theorems, or lemmas, in the proof produces a procedure, or subprogram, of the extracted program.

We had difficulty finding an existing theorem-proving system capable of carrying out the proofs of the theorems required. The Boyer-Moore theorem prover [BM79], for example, does not deal with full quantification (universal and existential) in the theorem to be proved; this is a serious obstacle to us because our synthesis theorems always have both quantifiers. The Argonne theorem prover [BLMO86], on the other hand, does not deal with mathematical induction, which is important to us for introducing recursive calls. Nuprl [Co86] applies to a purely constructive logic; we deal with a classical logic restricted only enough to allow us to extract programs from proofs.

## 1.2 Deductive Tableaux

We have developed a theorem-proving framework particularly well suited to the program synthesis application. In this framework, we manipulate a *deductive tableau* of *assertions* and *goals*, declarative sentences each associated with a term, called its *output entry*. While the assertions and goals of the tableau are all that we need for a pure theorem-proving task, the output entries are required for extracting a program from a proof.

The deduction rules of the framework introduce new assertions and goals with associated output entries, without changing the meaning of the tableau. These rules incorporate some of the most prominent theorem-proving techniques, including (non-clausal) resolution, mathematical induction, and term rewriting.

## 1.3 Powerful Deduction Rules

In our design of the deductive-tableau system, we have emphasized the development of powerful deduction rules, which may achieve in a single step what would require many smaller steps in a conventional formal system. Our deduction steps resemble the intuitive steps in an informal argument. Our proofs are considerably shorter than conventional formal proofs, and the search space is correspondingly contracted.

Part of this effort is to incorporate properties of the background theory into the deduction rules, rather than representing them declaratively as assertions. Certain properties, when expressed declaratively, tend to spawn numerous logical consequences that, while sound, have little bearing on the problem at hand. We prefer to incorporate such properties into the deduction rules, so they may be invoked only when appropriate.

A special unification algorithm is used by several deduction rules, to incorporate equational properties (see Siekmann [Si89]) and sort properties (see Meseguer, Goguen, and Smolka [MGS]). Properties of ordering relations, such as transitivity and monotonicity, are built into our own *special relation* rules [MW86]. Other properties may be built into existing deduction rules using *theory attachments*, as in Stickel's [St85] theory resolution rule.

Within the deductive-tableau framework, we and our associates have worked out the derivations of many programs in numerical [MW87a] and list-processing ([Tr89], [Na89]) domains. We have at times derived programs we would have been unlikely to discover by conventional methods. Most of this work has been done by hand, to test the adequacy of our deductive system. In the past year, a good deal of our effort (with the help of some of our colleagues and students) has been devoted to the implementation of the method. Our first implementation has been interactive.

## 1.4 Interactive Implementation

The user of our system provides a specification of the desired program and any properties of the background theory that are not already familiar to the system. He or she must choose among a selection of legal alternatives presented by the system on a screen. The system displays the consequences of the user's choice and presents a further selection of alternatives. When the proof is complete, the final program is extracted and may be used as a subprogram in the derivation of future programs.

Although poor choices on the part of the user may postpone the derivation of the final program, it can never cause an erroneous program to be constructed. The system can only produce programs that meet the user's specifications.

The system has been used for both research and educational purposes.

## 1.5 Applicative vs. Imperative Programs

The deductive-tableau framework was originally introduced for the synthesis of *applicative* programs, which return an output but do not produce any side effects. The major emphasis of the ONR project has been the extension of the deductive approach to the synthesis of *imperative* programs, which may alter data structures and produce other side effects as part of their intended behavior. This extension has then been applied to the related problem of planning. We shall first describe how the deductive approach has been extended to imperative programs; we then describe the planning application.

## 1.6 The Trouble with Situational Logic

We first attempted to use a *situational logic,* a theory in which the situation, a state of the computation, is an object that may be treated in the same way as the data objects of an applicative program. Situational logic was proposed by McCarthy [Mc68]; a variant was used by Green et al [Gr69]. In constructing an applicative program, we proved the existence of a suitable output data object; in constructing an imperative program, we first attempted to prove instead the existence of a suitable final state. In other words, the state was treated as just another data object that the program could manipulate. Operations had as arguments several data objects and a state. For example, $square(x, s)$ might denote the state that results when $x$ was set to $x^2$ in state $s$, and $positive(x, s)$ might test whether $x$ is positive in state $s$.

This representation seemed to work in the synthesis of straight-line imperative programs (with no conditional tests), but it was found to break down in the synthesis of conditional imperative programs. In an applicative program, it is quite possible to apply two different operations to the same data object; for example, we can compute both $a+1$ and $a-1$. But in an imperative program, we cannot apply two operations in the same state. For example, once we have set $x$ to $x^2$ in state $s$, we have destroyed state $s$; we can no longer test whether $x$ was positive in state $s$. Unfortunately, programs constructed by a naive application of situational logic do apply many operations in the same state: the expressions $square(x, s)$ and $positive(x, s)$ may both occur in the same program. This is possible because, in this approach, states such as $s$ occur explicitly in the program we extract.

## 1.7 Fluent Theory

In circumventing this problem, we have been led to devise *fluent theory,* a situational logic in which a class of operations, called *fluents,* are explicit objects. Let us first be more precise about fluent theory; later, we shall show why this circumvents the problem with conventional situational logic.

Fluents are defined only in terms of what they do. Executing a fluent $e$ in state $s$ *returns* a data object $s : e$ and *produces* a new state $s\,;e$.

In specifying an imperative program, we formulate a relation

$$\mathcal{Q}[d_o, s_o, d_f, s_f]$$

between the input object $d_o$, the initial state $s_o$, the output object $d_f$, and the final state $s_f$. To construct a program to meet this specification, we do not merely prove the existence of an output object and final state satisfying the specified conditions. We prove the existence of a fluent $e$ such that executing $e$ in the initial state $s_o$ will return an output object $s_o : e$ and produce a final state $s_o\,;\,e$ satisfying those conditions. In other words, we prove the theorem

$$(\forall d_o)(\exists e)(\forall s_o)\mathcal{Q}[d_o,\ s_o,\ s_o : e,\ s_o\,;\,e].$$

The proof is again restricted to be sufficiently constructive to indicate a method for finding the desired fluent $e$, and that method becomes the basis for the program to compute $e$, which we extract from the proof.

In this approach, the data object $d_o$ is regarded as an input to the extracted program, but the initial state $s_o$ is not. States do not occur explicitly in the program at all; the same fluent is supposed to work correctly in any initial state. For this reason, the program never performs two operations in the same state.

The same deductive-tableau framework we have used to derive applicative programs may now be applied to derive imperative programs, provided fluent theory is part of the background theory. We have applied this approach to the derivation of several imperative list-processing programs [MW87b], such as imperative list concatenation and reversal, that alter the pointer structure of their arguments in computing the desired results.

## 1.8 Planning and Database Applications

The close analogy between planning and imperative program synthesis has long been recognized. We may regard the world as a rather large data structure, and the actions in a plan as the operations of an imperative program. Constructing a plan to achieve a given goal may then be treated as a problem of constructing a program to meet a given specification.

Exploiting this analogy, we have applied our fluent theory to the solution of robotic and commonsense planning problems [MW87c]. This approach gives us clean solutions to some somewhat troublesome problems.

Most work in planning has been devoted to the construction of straight-line plans; it has avoided the formation of conditional tests and of any sort of repetition or looping. These constructs give the planning system a mechanism for dealing with uncertainty. Although it is generally acknowledged that conditionals and loops are important, there is a tendency for researchers to postpone considering them.

In programming (as well as planning) applications, there is little justification for concentrating on straight-line programs. We chose a deductive approach over a purely transformational approach largely because of the relative simplicity of the deductive methods for conditional formation (via case analysis) and recursive loop formation (via mathematical induction). These methods carry over directly into the planning domain.

Some problems that are resolved simply in a deductive framework involve taking into account the action of the agent. Certain actions (e.g., getting a roadmap) must be inserted into a plan simply for the purpose of acquiring information. We have found that the same notion of "sufficient constructiveness" we use to ensure that the programs we extract from proofs are executable may also be used to ensure that the agent has sufficient knowledge to follow the extracted plan.

Another application we have investigated is database management. A database may be regarded as a sort of world model. The problem of updating a world model while maintaining given constraints is a typical planning problem and has been approached (in collaboration with X. Qian, a Ph.D. student specializing in databases) as a problem of deduction in fluent theory [QW88], [Qi89]. A system for the synthesis of database transactions, based on the system described in Section 1.5, has been implemented.

## 1.9 The Frame Problem

A well-known obstacle to the application of situational logics to the solution of problems in planning and imperative program synthesis is the *frame problem*, the odious necessity to state explicitly in our theory whenever a given operation can have no effect on a given relation. This places a heavy burden on the person specifying the background theory, because, particularly in planning applications, most operations may be regarded as independent of most relations. Furthermore, the numerous *frame axioms* that express these properties may overload the strategic capacities of the system, because they have numerous irrelevant consequences.

There is an active body of research on *nonmonotonic reasoning* [Gi87], in which one acknowledges that the background theory and its deduction rules are only an approximation to the truth, and introduces the possibility of retracting deductions when discrepancies arise. In such a system, it is possible to state an overly general frame axiom, such as that no operation has any effect on any relation. Some consequences of this axiom will be false, but may be retracted when they are found to contradict other axioms or conclusions.

The field of nonmonotonic reasoning is still in flux, and there is no general agreement as to how to proceed. Although we benefit from the results of this research, we have not addressed these problems ourselves. We include a correct description of the world, including the frame axioms, in our background theory.

The second aspect of the frame problem, as we have remarked, is the strategic burden of dealing with the numerous consequences of the frame axioms. We have found it advantageous to build these frame properties into the deduction rules, rather than expressing them declaratively as assertions, just as we have done for transitivity and other troublesome properties. One may incorporate these properties into the special relation rules [MW86] or as theory attachments to other rules [St85]. The upshot is that frame properties will not be invoked unless they are appropriate.

## 1.10 The Logical Basis for Computer Programming

A good deal of our time has been devoted to the completion of the second (and final) volume of our book (with Zohar Manna), *The Logical Basis for Computer Programming*. This volume provides an elementary exposition of the deductive-tableau framework and its application. It also provides an exceptionally clear introduction to many of the topics basic to the understanding of automated deduction, including well-founded induction, skolemization, and unification. The book is published by Addison-Wesley; Volume II appeared in late 1989.

## 1.11 Current Status and Publications

With the help of our ONR support we have accomplished the following:

- Development of a modified situational logic, called fluent theory, for the derivation of imperative programs

- Adaptation of fluent theory to planning
- Extension of the deductive-tableau framework to produce fluent theory proofs
- Implementation of an interactive system to prove theorems and derive programs within the deductive-tableau framework
- Application of fluent theory to the derivation of database transaction programs
- Completion of Volume II of *The Logical Basis for Computer Programs*
- Preliminary design of a semiautomatic system for planning, imperative program synthesis, and theorem proving

## 2 Publications

All of our technical reports have been published in journals or conference proceedings. A paper on fluent theory and its application to imperative program synthesis, "The Deductive Synthesis of Imperative LISP Programs," was presented at the 1987 National Conference on Artificial Intelligence, and appeared in the proceedings [MW87b]. A description of the application of fluent theory to planning problems, "How to Clear a Block: A Theory of Plans," was presented (in parts) at various workshops and appears in the *Journal of Automated Reasoning* [MW87c].

A description of the application of fluent theory to database management [QW88], "A Transaction Logic for Database Specification," appears in the proceedings of SIGMOD'88. The work is described more fully in the Stanford University Ph.D. Thesis of Xiaolei Qian [Qi89]. A response [Wa87] to Drew McDermott's critique on deductive methods, "The Bomb in the Toilet," appears in *Computational Intelligence* (1987). A basic introduction to the deductive-tableau method and a typical example of its application, "The Origin of a Binary-Search Paradigm," appears in the journal *Science of Computer Programming* [MW87a]. *The Logical Basis for Computer Programming*, Volume II: *Deductive Systems*, was published by Addison-Wesley [MW90].

## 3 References

**[BLMO86** ] R. Butler, E. Lusk, W. McCune and R. Overbeck, Paths to High-Performance Theorem-Proving, *8th International Conference on Automated Deduction*, pp. 588–597, Springer-Verlag, 1986.

**[BM79** ] R.S. Boyer and J S. Moore, *A Computational Logic*, Academic Press, 1979.

**[Bu88** ] A. Bundy, The Use of Explicit Plans to Guide Inductive Proofs, *9th International Conference on Automated Deduction*, pp. 111–120, Springer-Verlag, 1988.

**[Co86** ] R.L. Constable et al., *Implementing Mathematics with the Nuprl Proof Development System,* Prentice-Hall, 1986.

**[Gi87** ] M.L. Ginsburg, editor, *Readings in Non-Monotonic Reasoning,* Morgan Kaufmann, 1987.

**[Gr69** ] C.C. Green, Application of Theorem Proving to Problem Solving, *International Joint Conference on Artificial Intelligence,* pp. 219–239, 1969.

**[MW80** ] Z. Manna and R. Waldinger, A Deductive Approach to Program Synthesis, *Science of Computer Programming,* 2(1):90–121, 1980; reprinted in C. Rich and R.C. Waters, editors, *Readings in Artificial Intelligence and Software Engineering,* pp. 3–34, Morgan Kaufmann, 1986.

**[MW86** ] Z. Manna and R. Waldinger, Special Relations in Automated Deduction, *Journal of the ACM,* 33(1):1–59, 1986.

**[MW87a** ] Z. Manna and R. Waldinger, The Origin of a Binary-Search Paradigm, *Science of Computer Programming,* 9:37–83, 1987.

**[MW87b** ] Z. Manna and R. Waldinger, The Deductive Synthesis of Imperative LISP Programs, *Sixth National Conference on Artificial Intelligence,* pp. 155-160, Morgan Kaufmann, 1987.

**[MW87c** ] Z. Manna and R. Waldinger, How to Clear a Block: A Theory of Plans, *Journal of Automated Reasoning,* 3:343-377, 1987.

**[MW90** ] Z. Manna and R. Waldinger, *The Logical Basis for Computer Programming,* Addison-Wesley, 1990.

**[Mc68** ] J. McCarthy, Situations, Actions, and Causal Laws, in M. Minsky, editor, *Semantic Information Processing,* pp. 410–417, MIT Press, 1968.

**[MGS** ] J. Meseguer, J. Goguen, and G. Smolka, Order-Sorted Unification, to appear in the *Journal of Symbolic Computation.*

[Na89 ] D. Nardi, Formal Synthesis of a Unification Algorithm by the Deductive-Tableau Method, to appear in the *Journal of Logic Programming.*

[Qi89 ] X. Qian, A Deductive Approach to Database Transactions, Ph.D. Dissertation, Department of Computer Science, Stanford University, 1989.

[QW88 ] X. Qian and R. Waldinger, A Transaction Logic for Database Specification, *Special Interest Group on Management of Data '88,* pp. 243–250, ACM, 1988.

[Si89 ] J.H. Siekmann, Unification Theory, *Journal of Symbolic Computation,* 7(3/4), pp.207–274, 1989.

[St85 ] M.E. Stickel, Automated Deduction by Theory Resolution, *Journal of Automated Reasoning,* 1(4):333–335, 1985.

[Tr89 ] J.C. Traugott, Deductive Synthesis of Sorting Programs, *Journal of Symbolic Computation,* 7(6):533–572, 1989.

[Wa77 ] R.J. Waldinger, Achieving Several Goals Simultaneously, in E.W. Elcock and D. Michie, editors, *Machine Intelligence 8: Machine Representations of Knowledge,* pp. 94–136, Ellis Horwood, 1977.

[Wa87 ] R. Waldinger, The Bomb in the Toilet, *Computational Intelligence,* 3(3):220–221, 1987.