



STANFORD RESEARCH INSTITUTE
Menlo Park, California 94025 · U.S.A.

Final Report

August 1973

Covering the Period 1 June 1971 to 31 July 1973

RESEARCH IN ADVANCED FORMAL THEOREM-PROVING TECHNIQUES

By: B. RAPHAEL, R. FIKES, and R. WALDINGER

Prepared for:

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION
HEADQUARTERS
600 INDEPENDENCE AVENUE, S.W.
ROOM 607
WASHINGTON, D.C. 20546
Attention: MR. CHARLES PONTIOUS

CONTRACT NASW-2086

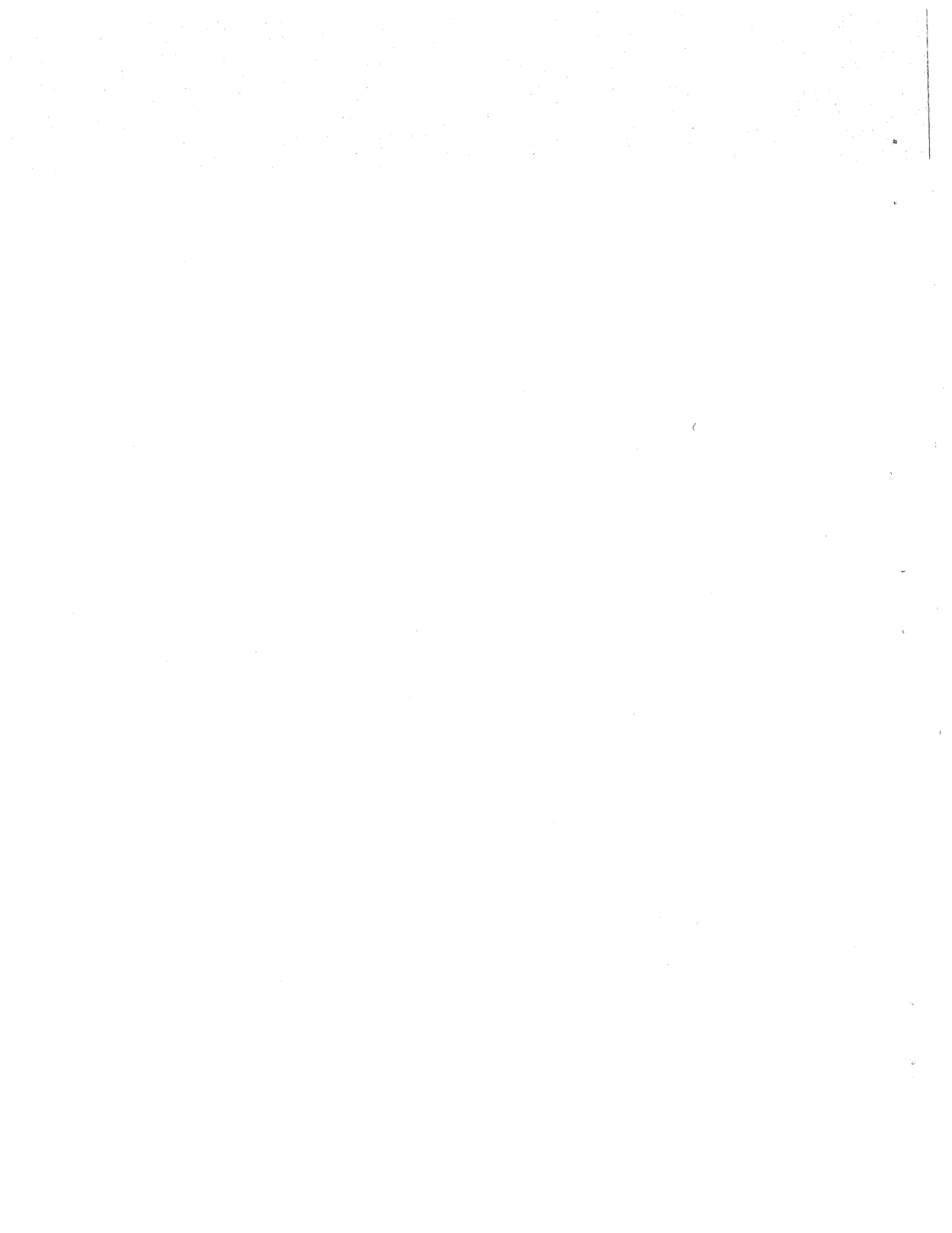
SRI Project 8721

Approved by:

B. RAPHAEL, *Director*
Artificial Intelligence Center

BONNAR COX, *Executive Director*
Information Science and Engineering Division

Copy No.¹⁰⁰



ABSTRACT

This report summarizes the results of a three-year project aimed at the design and implementation of computer languages to aid in expressing problem solving procedures in several areas of artificial intelligence including automatic programming, theorem proving, and robot planning. The principal results of the project have been the design and implementation of two complete systems, QA4 and QLISP, and their preliminary experimental use. QA4 has been documented in detail in a previous technical report.*¹ This report contains a description of how both QA4 and QLISP have been used; the Preliminary QLISP Manual is attached as an appendix.

*References are listed at the end of this report.

CONTENTS

ABSTRACT	iii
TABLE OF CONTENTS	v
LIST OF ILLUSTRATIONS	vii
ACKNOWLEDGMENTS	ix
I INTRODUCTION	1
A. Background	1
B. New Programming Languages for AI Research	2
C. Overview of the Project	4
II APPLICATIONS OF QA4 AND QLISP	5
A. A Deductive Retrieval System	5
B. A General Tree Searching Package	8
C. A Hierarchical Robot Planning and Execution System	10
D. Program Verification	12
REFERENCES	19
APPENDIX	21



ILLUSTRATIONS

1 Simplified Flow Chart for Perform 13

ACKNOWLEDGMENTS

No useful programming system can be developed without the cooperation of users who are willing to exercise each new experimental release, thereby helping to debug the system and provide valuable criticism to help in designing the next improved version. In addition to the authors of this report and its appendix, the following people helped us by using various undebugged versions of QA4 and QLISP: J. F. Rulifson, N. J. Nilsson, K. Levitt, B. Elspas, C. C. Green, I. Greif, M. Stickel, D. Lenat, D. Shaw, T. Garvey, and A. E. Robinson. The research applications for which QA4 and QLISP have been used are supported at SRI by the following sources: ARPA, under contracts DAHC04-72-C-0008 and DAHC04-72-C-0009; NSF, under grant GJ-eg146; and ONR, under contract N00014-71-C-0294.

I INTRODUCTION

A. Background

During the late 1960s the SRI Artificial Intelligence Center was engaged in a variety of research projects in the areas of automatic theorem proving,² question answering,³ problem solving,⁴ and robot systems.⁵ One of the themes of this collection of projects was the use of a theorem proving system, QA3,⁶ as the single deductive mechanism applicable to a range of other project goals. This approach had obvious appeal: Isolating the deductive component as an identifiable module simplified the conceptual structure of the various systems, and one could hope that improvements to the theorem proving module could cause the performance of all the other systems to improve simultaneously.

In practice, our approach had limited success. We did indeed develop demonstration systems for question answering and robot planning that were at least comparable in ability to any others in existence at the time. However, these systems could not be extended easily to handle larger, more realistic problem domains. A major bottleneck seemed to be the limited expressive power of first-order predicate calculus, the formal language used in our theorem proving system, for encoding the complex knowledge needed by the computer to solve harder problems.

In June, 1970, we began work for NASA on advanced formal theorem proving techniques. Our original plan was to develop a richer logical calculus to substitute for our QA3 system. We quickly discovered, however, that the problem of developing an automatic theorem prover for a system of logic--such as a higher-order predicate calculus that has the expressive power needed for interesting artificial intelligence research--

is itself a difficult problem of artificial intelligence (AI). Moreover, we decided that work on this class of problems was severely handicapped by inadequacies of the then-existing programming systems. Just as list-processing languages freed AI programmers from a mass of bookkeeping details and enabled a spurt of major research results a decade ago, we felt that new languages with a variety of novel built-in features we were just beginning to understand could promote a major step forward in AI progress today. Therefore, the NASA supported effort quickly turned from the study of theorem proving techniques per se to the development of software systems that we felt were prerequisites for major research progress in a broad spectrum of AI activities.

This report describes the results of three years' work on the development and experimental use of new programming languages for AI research.

B. New Programming Languages for AI Research

A new generation of programming languages is now becoming available. These languages, which have many features in common, were developed more or less simultaneously by several AI research laboratories, in response to largely independently-discovered needs. A tutorial review of some of these languages is given in Ref. 7.

The special features common to most of these languages can be described under the following headings.

- (1) Data Types and Memory Management. In addition to the now familiar symbolic data types, lists and strings, the new languages allow manipulation of such constructs as sets, trinary associations, and formal theorems. They also usually provide large, permanent data stores, with efficient built-in storage and retrieval procedures.

- (2) Control Structures. The major innovations are automatic backtracking, programmable changes of control environment ("contexts"), pseudoparallel processes, and automatic condition monitoring devices ("demons").
- (3) Pattern Matching. Complex pattern matching functions can be used for verifying the structure of the data, binding variables to subexpressions of the data, and selecting appropriate programs to execute.
- (4) Deductive Mechanisms. Built-in search and default inference procedures are useful features of these systems.
- (5) Special Operating Environment. Opportunity for intimate on-line interaction and powerful debugging facilities greatly enhance the effectiveness of a programming system.

The following languages are the major representatives of this new generation.

- (1) SAIL,⁸ a combination of ALGOL and an associative information retrieval system called LEAP, extended to meet the needs of the Stanford AI Laboratory.
- (2) PLANNER,⁹ a language developed at MIT in connection with the concept of procedural representation of knowledge.
- (3) CONNIVER,¹⁰ a successor of PLANNER that gives the programmer more direct control and responsibility with respect to program direction.
- (4) POPLAR,¹¹ an extension of the POP2 language developed at the University of Edinburgh School of AI.
- (5) QA4 and QLISP, the languages developed at SRI under the project reported here.

We believe that QLISP contains most of the desirable features of this new generation of languages in a well designed system framework, and that it has an excellent chance of becoming the most important software tool for the next few years' AI research.

C. Overview of the Project

During the first year of this project, 1970-71, our ideas about needed language features were crystallizing. These ideas were documented in a variety of memos and papers that are summarized in an annual report.¹² The second year, 1971-72, was primarily devoted to the implementation, refinement, and initial use of a major new programming language, QA4. That system, along with its motivation, philosophy, and uses, was documented in a 360-page technical report submitted in November of 1972.¹

Since late 1972, our primary activity has been the development of a second generation of our new language. The resulting system, called QLISP, will contain all the desirable features of QA4; but, they will be packaged in a much more efficient and more usable framework. We expect QLISP to be the major programming system for AI research at SRI for years to come.

Although QLISP still needs some significant modifications, a preliminary version is now operational. The Preliminary QLISP Manual is attached as an appendix to this report. We plan to make this manual available separately as a Technical Note in order to promote experimentation with the system.

Even though QA4 and QLISP are still incomplete experimental systems, we have already used them effectively for substantive research, partially supported by other projects, in the general areas of robot planning, theorem proving, and automatic programming. The next section of this report describes some of these applications.

II. APPLICATIONS OF QA4 AND QLISP

During the past year QA4 and QLISP have been used as tools in the exploration of design ideas. With the aid of QA4 and QLISP we have found it much simpler to create trial implementations of ideas and to run experiments with those implementations than we have in the past. QA4 and QLISP contain the pattern matching, expression manipulation, modeling, and control that are common to much of our work, and therefore a new program design idea can be quickly implemented without having to expend large amounts of effort rebuilding these basic mechanisms. Some of these implementations and features of QA4 or QLISP that were particularly useful are discussed below. This discussion assumes that the reader is familiar with the basic QA4 notations, such as inverse quote mode, and concepts, such as the use of "demons." These notations and concepts are described extensively in Ref. 1.

A. A Deductive Retrieval System

In our work with robot planning and more recently in our work with a system for providing guidance to a man doing some maintenance task, our programs need to maintain a model of the physical environment in which the robot activity or maintenance operation is taking place. Our planning, monitoring, and advice-giving programs ask questions of this model to determine how they should proceed. Hence, we need a question answering subsystem that can interface with the model. This subsystem should provide a deductive capability for dealing with questions whose answers are not explicitly stored in the model but whose answers can be deduced from information that is in the model. A new design for such a subsystem was tested during the last year with an implementation in QA4.

This design is an augmentation of the QA4 model retrieval statements EXISTS and INSTANCES. These statements retrieve from the model one or all true instances of a given expression containing variables (i.e., a pattern). For example, the statement (EXISTS (ON ←X DESK1)) could retrieve from the model one object that is on DESK1, and the statement INSTANCES (ON ←X DESK1) could retrieve all objects that are on DESK1. Now, consider as another example the statement (EXISTS (TOP:CLEAR DESK1)). This statement will look in the model for the expression (TOP:CLEAR DESK1) to indicate that no objects are on DESK1. If that expression does not have value TRUE in the model then the query will fail even though a simple deductive process that looked in the model for expressions of the form (ON ←X DESK1) could determine an answer to the query. The new design attempts to make it easy to add such deductive processes to the system and thereby turn the retrieval mechanism into a question answering mechanism.

The design allows the user to associate with each predicate in the system (e.g., ON, TOP:CLEAR, IN:ROOM) a list of deductive programs that will be called whenever true instances of an expression containing the predicate are being searched for in the model. Hence, one could associate a deductive program with the predicate TOP:CLEAR that would perform as described in the previous paragraph.

One common use of these deductive programs is to determine that an expression is false based on uniqueness properties of the predicate. For example, if there is a query asking whether OBl is in room Rm1, i.e., (EXISTS (IN:ROOM OBl Rm1)), a deductive program could look in the model for any true expression of the form (IN:ROOM OBl ←X). If such a true expression is found and the room name that matched with ←X is not Rm1, then the deduction can be made that OBl is not in Rm1.

Standard deduction programs are included in the system for NOT, AND, and OR. These programs allow one to ask questions such as: "Find

a desk in either room RM1 or RM2 whose top is clear." The QA4 form of that query might be (EXISTS (AND (TYPE ←X DESK) (OR (IN:ROOM ←X RM1) (IN:ROOM ←X RM2)) (TOP:CLEAR ←X))). To answer this query an AND deductive program is first called. It would ask for an object of type DESK from the model. If one is found, say D1, then it asks if (OR (IN:ROOM D1 RM1) (IN:ROOM D1 RM2)) is true. This query calls an OR deductive program which might in turn call an IN:ROOM deductive program that could determine which room D1 is in by asking the model for the location of D1. If D1 is found to be in either room RM1 or RM2, then the AND program will ask if (TOP:CLEAR D1) is true. This might cause a TOP:CLEAR deductive program to be called that would ask the model for any objects X for which (ON ←X D1) is true; if no such objects are found, then D1 is returned as a desk that satisfies the conditions in the original query.

This question answering process makes use of the QA4 backtracking capabilities when there is a need for more than one answer to a query. For instance, if our example query had been to find all the desks in either room RM1 or RM2 with clear tops, then each time such a desk was found it would be saved in a set and the answering process would be backtracked to a point in the deductive programs where alternative choices were available. In this example, control would return to the search for objects of type DESK, and, if another desk were found, the process would proceed as before to check the room the desk is in and whether its top is clear. Each desk would be checked in this manner to form the desired set.

The backtracking mechanism is also used within the answering process when, for example, a desk that is found by the (TYPE ←X DESK) query is not in either RM1 or RM2. In that case the OR deductive program returns FALSE to the AND deductive program and control is backtracked into the (TYPE ←X DESK) query to find the next desk.

